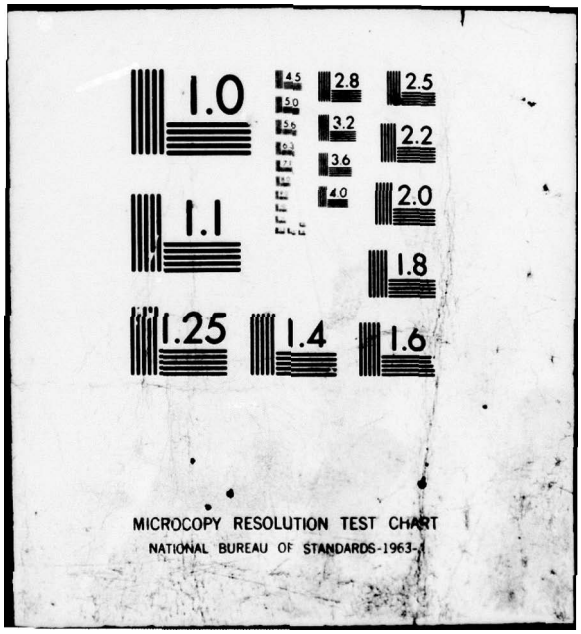
AD-A073 653 AMHERST SYSTEMS INC BUFFALO NY F/G 9/2
IMAGE PROCESSING SYSTEM SOFTWARE. VOLUME II. PROGRAMMING MANUAL--ETC(U)
JUN 79 E G EBERL, P T GLINSKI F30602-78-C-0077
UNCLASSIFIED AMHERST-0077-VOL-2 RADC-TR-79-52-VOL-2 NL

1 OF 3
AD A073653


1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
4.5
5.0
MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

(12) LEVEL III

AD A073653

RADC-TR-79-52, Vol II (of two)
Final Technical Report
June 1979

IMAGE PROCESSING SYSTEM SOFTWARE
Programming Manual

Amherst Systems, Inc.

Dr. Edward G. Eberl
Mr. Philip T. Glinski

ROME AIR DEVELOPMENT CENTER
Air Force Systems Command
Griffiss Air Force Base, New York 13441

IMAGE PROCESSING SYSTEM SOFTWARE, VOL II

This report has been reviewed by the RADC Information Office (OI) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

RADC-TR-79-52, Vol II (of two) has been reviewed and is approved for publication.

APPROVED: Anthony R. Fanelli

ANTHONY R. FANELLI
Project Engineer

APPROVED: Howard Davis

HOWARD DAVIS
Technical Director
Intelligence & Reconnaissance Division

FOR THE COMMANDER: John P. Huss

JOHN P. HUSS
Acting Chief, Plans Office

If your address has changed or if you wish to be removed from the RADC mailing list, or if the addressee is no longer employed by your organization, please notify RADC (IRRE) Griffiss AFB NY 13441. This will assist us in maintaining a current mailing list.

Do not return this copy. Retain or destroy.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER: RADC-TR-79-52, Vol II (of two)	2. GOVT ACCESSION NO.
3. RECIPIENT'S CATALOG NUMBER	
4. TITLE (and Subtitle): IMAGE PROCESSING SYSTEM SOFTWARE, Volume II. Programming Manual,	5. TYPE OF REPORT & PERIOD COVERED: Final Technical Report, Feb 78 - Nov 78,
6. PERFORMING ORG. REPORT NUMBER: Amherst - 0077-VOL-2	
7. AUTHOR(s): Dr. Edward G. Eberl; Mr. Philip T. Glinski	8. CONTRACT OR GRANT NUMBER(s): F30602-78-C-0077
9. PERFORMING ORGANIZATION NAME AND ADDRESS: Amherst Systems, Inc., 132 Cayuga Road, Buffalo NY 14225	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS: 62702F; 62441086
11. CONTROLLING OFFICE NAME AND ADDRESS: Rome Air Development Center (IRRE), Griffiss AFB NY 13441	12. REPORT DATE: Jun 1979
	13. NUMBER OF PAGES: 222
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office): Same (12) 216 p.	15. SECURITY CLASS. (of this report): UNCLASSIFIED
	15a. DECLASSIFICATION DOWNGRADING SCHEDULE: N/A

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

Same

18. SUPPLEMENTARY NOTES

RADC Project Engineer: Anthony R. Fanelli (IRRE)

DDC RECEIVED SEP 11 1979 B

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Image Processing
Interactive Processing
Computer Programming

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This document is the final report describing the effort by Amherst Systems, Inc. to convert a portion of the RADC image processing software to RSX-11M to allow execution on DEC PDP 11/45 computer located at WPAFB OH, AFAL. A section of this report is written in the form of a user's manual for personnel engaged in the operation of converted software. A companion document is the programmer's manual which describes the new material incorporated into the system.

DD FORM 1 JAN 73 1473

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

393 023

TABLE OF CONTENTS

TABLE OF CONTENTS (CONT.)

SECTION 1
SOFTWARE SYSTEM DESIGN

1.1 OVERALL SYSTEM ORGANIZATION

The image processing software is written to be executed on a PDP 11 computer as a program under the RSX-11M operating system. The image processing system is divided into 13 frames, each frame consisting of several related function options. Each frame is a unique task with its own control section. The control section allows selection of options within the frame and selection of a different frame. Options within a frame are overlayed using the RSX-11M autoload feature. If another frame is selected, the current frame execution is discontinued and the selected frame is loaded into memory and executed. Linked to each frame is a resident library containing commonly used IPS subroutines. The resident library is only loaded once and is simultaneously memory resident with the frame task.

The executable IPS software requires 17 files for system operation. They are 13 frame tasks, the Error Message file, the Master Option List file, the resident library task, and the device task. A complete set of command files exist which aid in the rebuilding of tasks from the source programs.

The IPS Software calls upon the services of RSX-11M to accomplish several functions. Frame options are overlayed using the autoload feature of the task builder. Data files on disk and tape peripherals are accessed through the file control services (FCS) of RSX-11M. Several other functions, such as time of day access, and communication with peripheral devices (as the Tektronix Terminal), are accomplished via the services of RSX-11M.

The system software can reside on any disk in the hardware configuration that is recognizable to RSX-11M. Data space must exist beyond that required for IPS Software to allow for the storage of IPS Data files.

The IPS Data files are created within the RSX-11M file structure. All software uses the RSX-11M file specification (device, unit, file name, extension, and version) when referencing a file. However, due to the desirability of maintaining compatability between the original IPS file format under DOS and the new RSX-11M version, all filenames should not exceed 6 characters and file version numbers

should not be specifed. The difference between the old DOS IPS and the new RSX-11M IPS is then transparent to the user.

1.2 THE SYSTEM FRAMES

There are 13 frames in the IPS Software. Each frame is a unique, executable task. A frame contains several functionally related options. Each frame has the capability of transferring control to any other frame installed in the system.

When a frame task begins execution, control is transferred to "FSTRT". "FSTRT" initializes the frame. The terminal number and task name are stored in common. The Error file and Master Option file are checked for their existence. If either is missing, a fatal error results. If the Log file exists, the Log is assumed to be on and it is appended to. Otherwise, the Log is considered to be off.

Control then transfers to "RSTRT" which is the frame restart routine. The stack pointer is reset and the display is rebuilt with the option list. This routine is be be entered whenever the frame is to be restarted. This occurs upon a fatal error.

"CTLOOP" is the Control Loop of the frame. "CTLOOP" inputs a user command and then takes appropriate action. There are four unique inputs which are recognized. If "EX" is entered, the frame exits to RSX-11M. If a decimal number is entered, this number identifies a selected option. A call to the selected option entry point is executed. When the option is finished, control is returned to "CTLOOP". If a blank line is entered, the master option list is displayed and a new frame number entered. Control is then transferred to "STRTSK". If an "F" is entered followed by a number, it is considered a new frame number.

"STRTSK" is the Start Task Routine. This Routine accepts the new frame number and starts execution of the indicated frame task. The current frame task is exited and the Log File is closed only if it exists.

1.3 FILE SYSTEM

The image processing file system is designed with the RSX-11M file structure. It is

essentially an interface between the option routines which access the files and the RSX-11M file functions which maintain the files and their associated directories. Special considerations have been incorporated into this interface to achieve rapid data access and to provide high-level file functions tailored to image processing needs.

All files are created as contiguous files. The contiguous file format allows several logical blocks of data to be transferred between disk and memory in one access. This has the effect of greatly reducing data-transfer time.

The internal structure of the data files is organized into fixed length records. The record length, as well as the number of records, is dependent upon the type of data stored within. Each file begins with a 256-word header, in which are stored various parameters describing the data. Detailed descriptions of these parameters can be found in Appendix E.

The data records immediately follow the header beginning in the first available logical device block after the header. Records immediately follow one another in consecutive order with no intervening spaces. This, of course, means that for certain record sizes, one or more records may cross device block boundaries at one or more points in the file.

A file is created or retrieved by calling the appropriate file system routine with an accompanying parameter list called a "request block". The format of this block is as follows:

WORD	DESCRIPTION
0	RSX-11M File Descriptor Block
96	Filename Prompt Address
97	Filename Prompt Length Address
98	Record Length
99	Number of Records in File
100	Linked Flag
101	Contiguous Records Requested
102	Record Number Requested
103	I/O Status Block

104	Buffer Size (Bytes)
105	Buffer Address
106	Blocks in Header
107	First Record in Buffer
108	Last Record in Buffer
109	Default Filename Block

The request block describes the file to be created or retrieved and the buffer space to be used for data I/O. This buffer space must be dedicated to the file until it is closed, at which time it is available for other use.

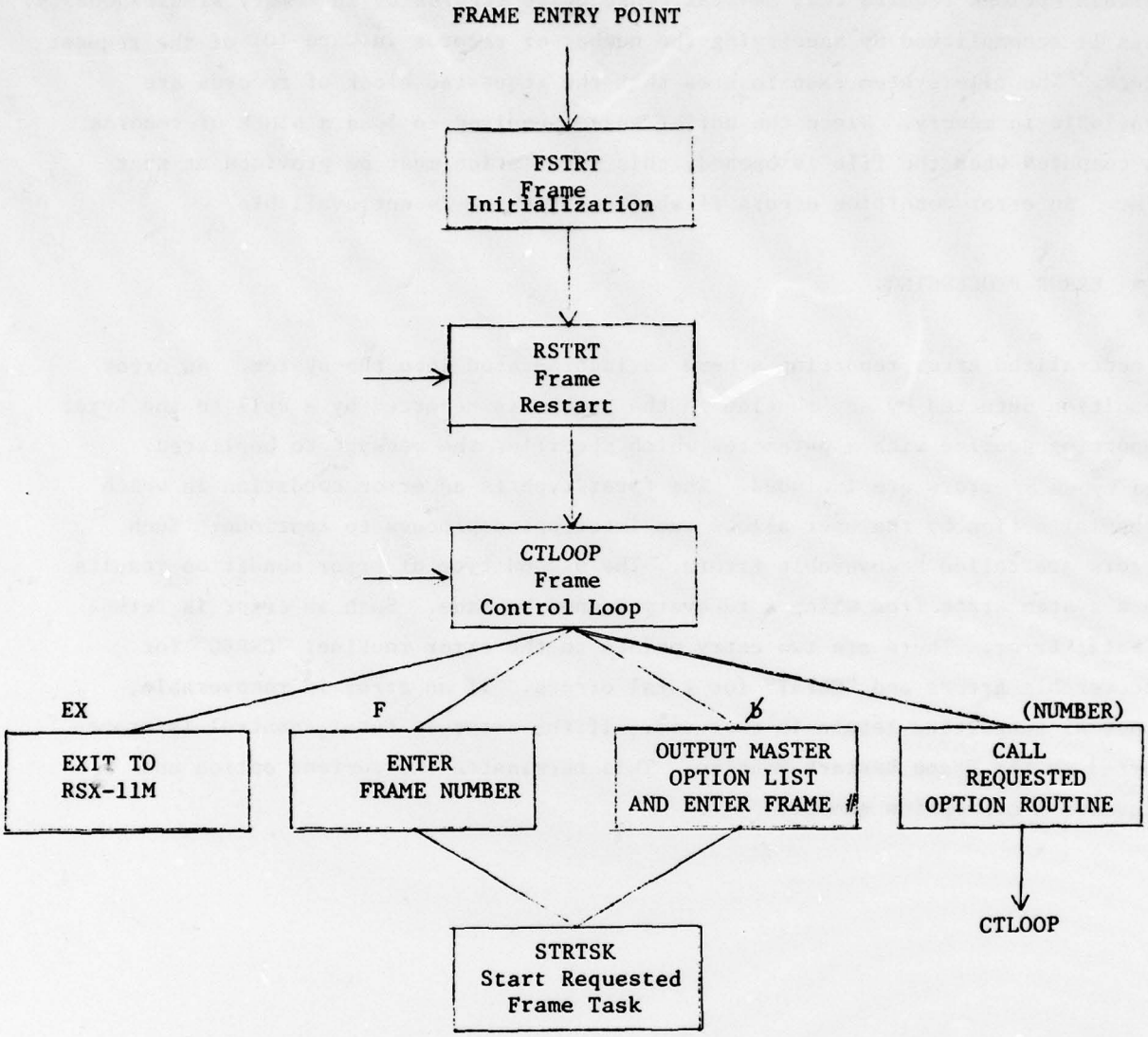
If the first word of the request block contains an address of a character string, the string is printed at the keyboard display as a query for a device and a file name. The user entry is then placed in the request block. If the character string address is not present, the file specification appearing in the request block is used. In either case, the file is created or retrieved as specified. The number of records and record length must be provided by the calling program for create operations and are returned by the filing system for retrieve operations. The options request data records by placing the number of the desired record into word 102 of the request block. The file system routines then read this record into memory for the option. Due to the typically sequential access of image files during processing, the file system is designed to anticipate the next records that will be requested by the options. Therefore, instead of reading just one record, the entire buffer specified in the request block is filled with records that immediately follow the requested record. The result is that the next request does not require a disk access since the record has already been obtained in a previous access.

A similar procedure is followed when writing data onto a file. For this operation, the specified buffer is filled prior to transfer to its disk file. However, sequential file output is not only assumed in this case, it is expected. Due to the complexities involved in providing random access when records can cross logical block boundaries, this capability has not been included. For options that require such access, a read/write mode is available. In this mode the buffer contents are transferred to the file and then the records beginning with the requested record are read into memory. This eliminates the problem of preserving the overlapping record data.

Certain options require that several consecutive records be in memory simultaneously. This is accomplished by specifying the number of records in word 101 of the request block. The file system then insures that the requested block of records are available in memory. Since the buffer space required to load a block of records is computed when the file is opened, this information must be provided at that time. An error condition occurs if sufficient space is not available.

1.4 ERROR PROCESSING

A centralized error reporting scheme is incorporated into the system. An error condition detected by any routine in the system is reported by a call to the Error Reporting Routine with a parameter which specifies the message to be listed. Two types of errors are included. The first type is an error condition in which remedial action by the user allows the interrupted process to continue. Such errors are called Recoverable Errors. The second type of error condition results in a system state from which a recovery cannot be made. Such an error is termed a Fatal Error. There are two entry points to the error routine; "ERREC" for Recoverable Errors and "ERFAT" for fatal errors. If an error is recoverable, a normal subroutine return is executed. If the error is fatal, control is transferred to the Frame Restart Routine. This terminates the current option and requests a new option number.



FLOW DIAGRAM OF A SINGLE FRAME TASK

SECTION 2

PROGRAMMER'S MANUAL

This section is intended as a programming guide for developing new software for the image processing system. A variety of services are offered by the executive, the file system and several utility programs. A large number of these services can be conveniently referenced via a set of system macros.
Although in some cases the programmer may find that directly inserting the code is just as convenient as using a macro, this should be avoided. By consistent use of the macros, modifications or updates can be made to an entire system by simply changing a few macros. Without the use of macros, a time-consuming search must be made of all routines in the system for occurrences of the code sequences being changed.

In the remainder of this section, several programming conventions and callable subroutines are discussed. Sufficient information is provided in each case to properly use these subroutines. Further details may be found in the program documentation in Section 4. For this purpose, the name of the Program in which the subroutine is contained is given here.

2.1 SUBROUTINE CALLING CONVENTIONS

All subroutine calls follow the general format given below:

```
     MOV        #1$, R5
     JSR        PC, (subroutine name)
1$:  BR         (offset to first instruction following the parameter list)
     Parameter list
                .
                .
                .
```

This format has two advantages. First, by including the branch instruction, the offset (lower byte) of the branch gives the number of parameters. For routines that accept variable length lists, the number of parameters passed can readily be determined.

The second advantage is that RSX-11M FORTRAN uses this calling format. Therefore, if the parameters are referenced in the list by their addresses, the routine is FORTRAN callable.

A general system macro has been provided to generate the proper subroutine calls. The format of this macro is as follows:

```
     call      name, P1, P2, P3, P4, P5, P6, P7, P8, P9, P10
```

which expands to

```
     MOV       R5,-(SP)
     MOV       #X$$A, R5
     JSR       PC, name
     MOV       (SP)+,R5
     .PSECT          .ARGS
X$$A:.WORD     #Of Parameters
     .WORD     P1
     .WORD     P2
     .WORD     P3
     .WORD     P4
     .WORD     P5
     .WORD     P6
     .WORD     P7
     .WORD     P8
     .WORD     P9
     .WORD     P10
     .PSECT
```

The number of parameters is optional within the range of 0 to 10. If no parameters are specified, only the JSR PC, name statement is generated.

The CALL macro should be used in all cases where a specialized macro is not available. In the subroutine descriptions to follow, the proper macro to be used in each case is discussed.

2.2 GRAPHICS DISPLAY INTERACTION

The graphics display is the prime means of communication with the system. This communication is accomplished via frame displays and user/system dialogue. Since this is a storage display, it must be erased and rebuilt from time to time to avoid overwriting of information. This rebuild action is under the control of the program "TELEIO" which together with "PLOT" and "TTYIO" contains the display I/O subroutines. Therefore, it is imperative that the user reference these subroutines for all display operations.

Since this is an interactive system, the programmer must frequently make requests of the user and accept his responses. These services are provided by the system at two levels. The higher-level routines are those which output a message and input the user's response. These consist of routines to request file specifications and numeric input. The lower level routines simply output or input character strings. These should not be used if a higher-level routine will satisfy the requirement.

Character strings consist of ASCII characters. Embedded carriage returns and line feeds are permitted. For normal user communications the line length should not exceed 40 characters. If a longer message is desirable, it should be sectioned into two or more lines with embedded carriage returns and line feeds. It should be noted that one complete message <u>should not</u> be output in sections by making several calls to the output routines. The initial lines of text may be lost if the display is rebuilt prior to completing the output. If the complete message is formatted with carriage returns and line feeds, then the output routines can insure sufficient display space for the entire message. (Only the routine "TTYOUT" detects imbedded line feeds. Input routines which issue a prompt, such as "TTYIN" do not. Therefore, inputs with multiple line prompts should be preceded by a call to "TTYOUT" which will output the first N-1 lines of the prompt. This will insure Proper Display Rebuild).

The available communication subroutines and their associated calling sequences are discussed below. The "CALL" macro is used in all cases to form the subroutine calling sequence.

2.2.1 Request File Specification

```
CALL    GETNAM, EXBUF, CHR, LEN, FRB, IND (Ref. program TTYIO)
```

where

```
EXBUF   = EXECUTIVE COMMON
CHR     = ADDRESS OF PROMPT
LEN     = ADDRESS OF LENGTH OF PROMPT
FRB     = ADDRESS OF FILE REQUEST BLOCK
IND     = ADDRESS OF BLANK LINE INDICATOR
```

The file name is input from the keyboard and parsed. The parsed file name information is returned in the file request block specified. If the input cannot be parsed, the input request is repeated. The device defaults to that assigned to pseudo device "IP". The UIC defaults to that which the user is presently running under. The version number defaults to one.

2.2.2 Request a String of Signed Double Word Integer Numbers

To input decimal numbers:

```
CALL    DBLDEC, EXBUF, CHR, LEN, CNT, BUF, IND  (IND is optional)
        (Ref. Program TTYIO)
```

To input octal numbers:

```
CALL    DBLOCT, EXBUF, CHR, LEN, CNT, BUF, IND  (IND is optional)
        (Ref. Program TTYIO)
```

where

CHR = Address of the output character string

LEN = Address of the length of the character string

CNT = Address of a location containing the count of numbers to input. This count must be satisfied exactly or an error message is printed and a request is made to retype the line.

BUF = Address of buffer in which to return the numbers in the order entered. This buffer must contain sufficient space to store all the numbers indicated by the second parameter. Double word values are returned low order first followed by high order.

IND = (Optional parameter) Address of a location in which to return an indicator. A negative one is returned if a CTRL/Z if found to be the first character entered by the user. Otherwise, a zero is returned.

This routine is designed to allow one or more numbers to be entered on one or more lines by the user. For multiple line input where the total number of lines is determined by the user, the call to DBLDEC should be contained within a loop. This call should use the optional fourth parameter "IND". The loop is exited when IND = -1, which occurs when the user indicates the end of input by responding with a CTRL/Z.

2.2.3 Request a String of Signed Word Integer Numbers

To input decimal numbers:

```
CALL    SNGDEC, EXBUF, CHR, LEN, CNT, BUF, IND  (IND is optional)
        (Ref. Program TTYIO)
```

The parameters and their descriptions are identical to that given for DBLDEC except that word values are returned in the buffer.

2.2.4 Request a String of Signed Byte Integer Numbers

To input decimal numbers:

```
CALL    BYTDEC, EXBUF, CHR, LEN, CNT, BUF, IND  (IND is optional)
        (Ref. Program TTYIP)
```

To input octal numbers:

```
CALL    BYTOCT, EXBUF, CHR, LEN, CNT, BUF, IND  (IND is optional)
        (Ref. Program TTYIO)
```

The parameters and their descriptions are identical to that given for DBLDEC except that byte values are returned in the buffer.

2.2.5 Request a String of Floating Point Numbers

```
CALL    FLTENT, EXBUF, CHR, LEN, CNT, BUF, IND  (IND is optional)
```

The parameters and their descriptions are identical to DBLDEC except two-word floating point numbers are returned. Numbers are accepted in either of the following formats:

```
19678.66
1.967866E04
```

where both entries represent the same number

2.2.6 Output a Character String

```
CALL    TTYOUT, EXBUF, CHR, LEN
        (Ref. Program TELEIO)
```

where

CHR = Address of the output character string. TTYOUT will append a final carriage return and line feed.

LEN = Address of the length of the character string

2.2.7 Input a Character String

```
CALL    TTYIN, EXBUF, CHR, LEN
        (Ref. Program TELEIO)
```

where:

EXBUF = Executive Common
CHR = Address of the output character string
LEN = Address of the string length

If an "ALT MODE" key is entered, input terminates and the equivalent of a fatal error is executed.

The input string is returned in the 80 byte buffer TTYBF$, located in "EXBUF". The input string length is returned in IOSLN$, located in "EXBUF".

2.2.8 Specialized Graphic Displays

Certain functions require that the graphics terminal be used for special display purposes. To gain control of the display, the following instruction should be executed:

```
CLR     REBLD$
```

This disables the display rebuild software. The display can then be manipulated by using the graphic plot subroutines below in addition to any of the other terminal I/O subroutines. After the specialized display mode is no longer required, the following instruction should be executed to enable the display rebuild software:

```
MOV     #1, REBLD$
```

The current frame is then redisplayed by the following call:

```
MOV     #EXBUF, R]
CALL    BLDISP
        (Ref. Program BLDISP)
```

Another special display feature is available to the programmer. This allows specialized displays to be presented and user/system dialogue to be maintained. To accomplish this, the address of a programmer-supplied routine that builds the specialized display must be placed in the global location "DSPDR$".

1. CLR REBLD$
2. Call the routine that presents the special graphics display. space must be reserved along the left display margin for dialogue. This routine must leave the alpha cursor positioned at the top of the left margin.
3. MOV #1, REBLD$

The dialogue can now be maintained with the system. When the dialogue reaches the bottom of the left margin, the control software will call the routine whose address appears at "DSPDR$" to rebuild the display. Dialogue then continues.

Upon exiting the special display mode, the "BLDISP" routine should be called and the address "BLDISP" should be replaced in "DSPDR$" to allow normal frame rebuild.

The following is a collection of routines to allow graphic plots to be constructed on the display terminal. The display manual should be consulted for complete descriptions of the various plotting operations. The calling sequences are as follows (all subroutines are in program PLOT):

To put the display in alpha mode:

```
CALL    ALPHA, EXBUF
```

To put the display in graphics mode:

```
CALL    GRMODE, EXBUF
```

To clear the display screen:

```
CALL    CLEAR, EXBUF
```

The display is left in alpha mode with the cursor at the top of the left margin.

To put the cursor in the home position (top left margin):

```
CALL    HOME, EXBUF
```

Following this call the display is in graphics mode:

To draw a light or dark vector:

```
CALL    PLOT, EXBUF, X, Y
```

where

EXBUF = Executive Common

X = The address of a location containing the X display coordinate

Y = The address of a location containing the Y display coordinate

A dark vector is drawn if the call is immediately preceeded by a call to "GRMODE". Successive calls to "PLOT" draw light vectors between the specified consecutive points.

To output a graphics string:

```
CALL    TTYGRF, EXBUF, CHR, LEN
```

where

EXBUF = Executive Common

CHR = Address of the output character string

LEN = Address of length of string

TTYGRF outputs all character codes in a write-through mode. No control character processing is performed by the driver.

2.3 FILE SYSTEM SERVICES

Extensive services are provided to the programmer for manipulating files. These services include create and retrieve functions. Accompanying the create and retrieve operations is the ability to read and/or write a file. A specialized parameter list called a "request block" has been developed for the create and retrieve functions.

2.3.1 Request Block Format

The request block is generated via the macro "F.REQ" and takes on two general forms - one for the create function and one for the retrieve function. For the create function the macro is invoked as follows:

```
F.REQ    CRE, LBL, CHR, CLN, EXT, RECS, LNG, ACES, FTYP, RTYP, NAME
```

The macro call for generating the request block for the retrieve function is:

```
F.REQ     RET, LBL, CHR, CLN, EXT, ACES, NAME
```

This expands in the same manner as for the create call with the exception that the parameters RECS, LNG, FTYP, and RTYP, which are not specified, are automatically set to zero. The definition of each of the parameters is as follows: (values of symbols as RD, WR, or CONTIG are defined by the Equate Macro)

LBL = One to three characters to be used as the first part of the parameter labels.

CHR = Address of the character string to be output to request the file specification. A zero value implies that there is no character string and that the file specification is completely provided in the request block. Therefore, no user interaction will result.

CLN = Address of the length of the above character string.

EXT = File extension which indicates the file type with respect to the data contained therein. If this argument is left blank or if the program clears the location associated with this argument, then the filing system uses the extension specified by the user. If the extension is specified via the macro argument or the program, then the user-entered extension is ignored.

RECS = The total number of records to be in the file.

LNG = The length of each record in bytes (odd values are acceptable)

ACES = File access type:

RD = Read

WR = Write

MO = Modify (read and write)

FTYP = File type with respect to structure:

CONTIG= Contiguous

LINKED= Linked (currently not implemented)

RTYP = Record type:

FLEN = Fixed length

VLEN = Variable length (currently not implemented)

NAME = File name. If the name is to be entered by the user at execution time or if the name is to be inserted by the program, this parameter should be left blank. If the name is known at programming time, it should be inserted. If temporary files are to be created, the .TMP extension should be used. This file will be deleted when the file is closed.

All request block parameters should be referenced by their respective labels. Future system changes or additions may result in a different order or number of parameters. Therefore, if in a given instance one parameter must be referenced relative to another, then the values defined by the macro "EQUATE" should be used. These values are simply indexes into the request block relative to the first parameter in the block. They are referenced via the last three characters of the label of their respective parameters. For example, if the buffer length parameter must be referenced with respect to the buffer address parameter, then the programmer would use.

```
LBLBUF+SIZ-BUF
```

as the relative address

2.3.2 Creating Files

Two file system subroutines are available for creating files. The only difference between them is that one creates the file and then opens it for access while the other simply creates the file. The request block parameters must be initialized prior to either call as discussed above. The one parameter which is of no concern at this point is the record number requested (LBLREC). This is used by other file functions. Special note should be taken of the file system feature which requests the file name from the user. This option is enabled or disabled as explained for the character string address (LBLCHR).

To create a file and open it for access:

```
F.CRE     ADR, ALTRET
          (Ref. Program is FILE1)
```

where

ADR = Address of the applicable request block.

ALTRET = Address of an alternate return to be taken if the user responds with a carriage return to the request for a file name. This parameter is optional and need not be specified. When it is not specified a carriage return alone is reported as a recoverable error.

To create a file (file is not opened):

```
F.CRE$    ADR, ALTRET
          (Ref. Program is FILE1)
```

where the parameters are as defined for the F.CRE macro.

Both of the above macros generate the appropriate subroutine call to the file system. After each call R0 contains the memory address of the file header. If any data is to be inserted into the header, it should be done at this point (the file access must be set for write or read/write). The first file access (see 2.3.4 below)causes the header to be written into the disk file and to be removed from memory. For the F.CRE$ call, the header is provided for reference only (the header is written into the file prior to return).

It should be noted that the buffer specified in the request block following an F.CRE call is initialized for data I/O and should not be modified. This buffer is not used for the F.CRE$ call. Instead, the header is loaded into a buffer in the executive common. This buffer is overwritten by the next F.CRE$, F.RET$, OR F.CHK call.

2.3.3 Retrieving Files

Files are retrieved in a manner similar to that explained for the create option above. Again two options are available, differing only by the "open" operation. The request block is prepared as instructed above. In this case, the number of records, the record length and the file and record types are return parameters. The instructions for the other parameters are the same. The option is also available to have the file system request the file name from the user.

To retrieve a file and open it for access:

```
F.RET     ADR, ALTRET
          (Ref. Program is FILE1)
```

where the parameters are identical to those in the F.CRE call above.

To retrieve a file (file is not opened):

```
F.RET$    ADR,ALTRET
          (Ref. Program is FILE1)
```

where the parameters are identical to those in the F.CRE call above.

Operations with reference to the file header and the specified buffer are as explained for the create operations. In this case, the header is provided only for reference following the F.RET$ call. The buffer is used by F.RET but not by F.RET$.

2.3.4 Accessing File Data

Following a create or a retrieve function that also opens a file, the file may be accessed via the file system. Prior to each call for a record or group of records, the record requested parameter "LBLREC" should be set to the desired record. The first record in a file is record 1. The calling sequence to request a record or group of records is:

```
F.PTR    ADR, END
         (Ref. Program is FILE2)
```

where

ADR = Address of applicable request block.

END = Address of a location to which control is to be returned if one or more of the records requested is non-existent. If omitted, the filing system will report a fatal error (E1.10) when a non-existent record is encountered.

Following each call to F.PTR, memory addresses of the first byte of each record requested are found on the stack. For example, if the programmer requests one record (parameter LBLCON), the address of the first byte of that record is found on top of the stack. If three records are requested, then three addresses are returned. In this case the address of the first of the three records is found on top followed, in order, by the other two. The number of consecutive records can be set ao any number but must not exceed the value specified at the time that the file was opened.

2.3.5 Check for File Existence

Certain operations simply require that a file's existence be verified or that its parameters be retrieved. The following routine performs these functions. The request block is initialized as for file retrieval. The call is:

```
F.CHK    ADR, NOFILE
         (Ref. Program is FILE1)
```

where

ADR = Address of the applicable request block.

NOFILE = Address of an alternate return if the file does not exist.

If the file exists, its number of records and record length are returned in the request block. Also, R0 contains the memory address of the file header. As

with F.CRE$ and F.RET$, the executive buffer is used instead of the buffer specified in the request block. Therefore, the desired header information should be extracted prior to any similar operation.

2.3.6 Delete Functions

The delete function used is the standard RSX-11M delete macro. The following will delete a file:

```
DELET$    #FRB
```

where:

FRB = Address of file request block

2.3.7 Closing Files

All files that have been opened must be closed following processing. Two calls are provided for this function. First, a single file may be closed via the following call:

```
F.CLOS    ADR
          (Ref. Program FILE3)
```

where

ADR = Address of the applicable request block.

A second available call will close all files that are currently open:

```
F.SHUT
          (Ref. Program FILE3)
```

If control is returned to the executive due to a fatal error occurring during program execution, any open files will be automatically closed. Any such files that were open for write or modify access will be marked in the fifth word of the header as having been prematurely closed. Subsequent accesses to these files will cause the filing system to warn the user of the file state and allow him to select an alternate file.

2.3.8 Extend a File

A function is provided to allow the length of a contiguous file to be extended. This function should only be used when necessary because it must create a new file, copy the old file to the new and delete the old file. This can be a time-consuming process if the file is large. When required, it is called as follows:

```
F.EXT     ADR, EXTADR
```

where

ADR = Address of the request block for the file

EXTADR = Address of a location containing the size of the extention in 256 word blocks.

Note: The file must be in a closed state.

2.4 CORE RESIDENT BUFFERS AND PARAMETERS

Certain buffers and parameters are made available within the executive common, thus avoiding the need to define them in each overlay. The main working buffer for file I/O begins at the address stored at location "FRCOR$". The size of this buffer in bytes is stored at location "FRLEN$."

```
INNAM:    .ASCIZ   /ENTER INPUT IMAGE NAME=/
```

and

```
OUTNAM:   .ASCIZ   /ENTER OUTPUT IMAGE NAME=/
```

Since the frequency of access of other file types is much lower, character strings have not been provided.

2.5 ERROR REPORTING

A centralized error reporting scheme has been adopted in the system. Errors are reported by the programmer via a subroutine call accompanied by the desired error class and error number. Two classes of errors are provided - fatal and recoverable. Fatal errors are those which make it impossible for the program to continue. When such an error call is made, the error reporting routine prints the specified message and passes control to the executive. Therefore, the programmer should not provide any return code.

Recoverable errors are those from which a recovery can be made, such as the user retyping an input parameter. In this case, control is returned to the calling program after the error is printed.

Within each error class exists a list of error messages. Each message is referenced by its corresponding number. The programmer simply passes the number of the error to the error reporting routines. The list of errors can be found in Appendix D.

2.5.1 Fatal Error Reporting

```
ER.FAT    NUM
          (Ref. Program EXEC)
```

where

NUM = The error number within the fatal error class.

2.5.2 Recoverable Error Reporting

```
ER.REC    NUM
          (Ref. Program EXEC)
```

where

NUM = The error number within the recoverable error class.

2.6 UTILITY ROUTINES

A variety of utility routines are available to support the programmer. The function of and access to each routine is discussed below:

2.6.1 Converting Double Word Binary to Decimal ASCII

A routine is available to convert double word binary numbers to decimal ASCII character strings. Two modes of access are available. One mode passes all parameters in general registers and the other mode passes them in a parameter list following the subroutine call. The ASCII string is returned within twelve bytes, right justified, with leading zeroes blank. The minus sign, if present, is right justified with the number. After conversion the number of non-blank characters returned is available at the location "DIGCNT". The register mode call is as follows:

```
CALL      DI2DAR
          (Ref. Program CB2DA)
```

The following registers must be loaded prior to the call:

R2 = Address of the buffer in which to return the ASCII string.

R3 = High order part of double word value.

R5 = Low order part of double word value.

R1 = Address of Executive Common (EXBUF)

Parameter list mode:

```
CALL    DI2DA, DBLINT, BUFADR, EXBUF
        (Ref. Program CB2DA)
```

where

DBLINT = Address of double word integer (low order part first).

BUFADR = Address of a twelve-byte buffer in which to return the ASCII string.

EXBUF = Address of Executive Common

2.6.2 Converting Single Word Binary to Decimal ASCII

Single word binary conversion is also available. As with double word conversion, two modes are available for passing parameters. One is via general registers and the other is via a parameter list following the subroutine call. The ASCII string is returned within seven bytes, right justified, with leading zeroes blank. The minus sign, if present is right justified with the number. After conversion, the number of non-blank characters returned is available at the location "BKIND$". The register mode call is as follows:

```
CALL    SI2DAR
        (Ref. Program CB2DA)
```

The following registers must be loaded prior to the call:

R2 = Address of the buffer in which to return the ASCII string

R4 = Binary value to be converted.

R1 = EXBUF

Parameter list mode:

```
CALL    SI2DA, SNGINT, BUFADR, EXBUF
        (Ref. Program CB2DA)
```

where

SNGINT = Address of single word integer.

BUFADR = Address of a seven-byte buffer in which to return the ASCII string.

EXBUF = Address of Executive Common.

2.6.3 Converting Floating Point to ASCII

Two conversion methods are available which correspond in function to the FORTRAN "F" and "E" formats. The "F" format returns a number with the decimal point

fixed in its proper position. The "E" format returns a number with a decimal point plus a power of ten. The calling sequences follow (due to the .BYTE parameters, the CALL macro may be inconvenient):

F format conversion:

```
CALL    FPASCF, EXBUF, W, D, ARG, BUF
        (Ref. Program FPASC)
```

where

EXBUF = Address of Executive Common.

W = Address of the width of the field of characters to be returned.

D = Address of the number digits desired to the right of the decimal point.

ARG = Address of the two-word floating point number.

BUF - Address of the buffer in which the characters are to be returned.

E format conversion

```
CALL    FPASCE, EXBUF, W, D, ARG, BUF
        (Ref. Program FPASC)
```

(The parameters and their descriptions are identical to those for FPASCF.)

If the format is unacceptalbe, all characters in the return buffer are set to *'s. In "E" format, two *'s are returned for the exponent if it is greater than 99. Either error condition causes the "C" bit in the processor status register to be set. This bit is otherwise cleared.

2.6.4 Convert Radix 50 Packed Characters to ASCII

This routine converts radix 50 packed characters to ASCII. The call is as follows:

```
CALL    RAD2AS, RAD50, ASCII, COUNT
        (Ref. Program RAD2AS)
```

where

RAD50 = Address of the first word in the radix 50 packed string of words.

ASCII = Address of a buffer in which the ASCII string will be returned.

COUNT = Address of the number of radix 50 packed words.

2.6.5 Save and Restore General Registers

This routine saves and restores general registers R0 through R5 using the stack. No parameters are passed.

Saving registers:

SAVREG
(Ref. Program SAVER)

Restoring registers:

RSTREG
(Ref. Program SAVER)

2.6.6 Save and Restore Floating Point Registers

This routine saves and restores the floating point registers AC0 through AC5, the floating point processor interrupt vector, and the floating point status register using the stack. No parameters are passed.

Save FPU status:

SAVFPS
(Ref. Program SARFPS)

2.6.7 Square Root of a Double Word Integer

This routine calculates the square root of a double word integer. All registers are unaffected by this routine. The call is:

```
CALL    DPSQRT, DBLINT
        (Ref. Program SQRT)
```

where

DBLINT = Address of the double word integer (low order first). Upon return the square root is found in the first word of the double word. The second word is zero.

2.6.8 Square Root of a Single Word Integer

This routine calculates the square root of a single word integer. All registers are unaffected by this routine. The call is:

```
CALL    SQRT, SNGINT
        (Ref. Program SQRT)
```

where

SNGINT = Address of the single word integer. Upon return this location contains the square root.

2.6.9 Partitioning Core

This routine partitions the FRCOR$ buffer into specified fractional parts and loads the address and size of each partition into their corresponding file request blocks. The call is (a macro is not provided):

```
     MOV       #1$, R5
     JSR       PC, PARCOR
1$:  BR        .+(n+1)*4
     .WORD     EXBUF
     .WORD     DENOM
     .WORD     NUMER1
     .WORD     BUFAD1
     .WORD     NUMER2
     .WORD     BUFAD2
          .
          .
          .
     .WORD     NUMERn
     .WORD     BUFADn
```

where

EXBUF = Address of Executive Common.

DENOM = The denominator of the fractional partitions. Each partition is expressed as a fraction of the total; therefore, the denominator is the same for all partitions.

NUMER1-NUMERn= The numerators for expressing the size of the n partitions.

BUFAD1-BUFADn= The n addresses of the buffer address parameters in the respective request blocks.

2.7 MISC. MACROS

This section discusses a number of macros which may be useful to the programmer. Use of these macros is at the option of the programmer with the exception of the EQUATE macro. This macro should always be used when the services it provides are required.

2.7.1 Partitioning Core Buffers

This macro partitions the specified core area into two equal-length buffers and deposits the start address and size of each in their respective request blocks. The user specifies the core space which is to be partitioned. The "FRCOR$" buffer is used as a default value.

```
CORD2   ADDR1, ADDR2, NOTLOW
```

which expands to

```
EQUATE                          (macro as defined below)
MOV     FRCOR$, R4
MOV     FRLEN$, R3              (included only if NOTLOW is blank)
CLC
ROR     R3
BIC     #1, R3
MOV     R4, ADDR1
MOV     R3, ADDR1+SIZ-BUF
ADD     R3, R4
MOV     R4, ADDR2
MOV     R3, ADDR2+SIZ-BUF
```

where

ADDR1 = The address of the buffer address parameter within the first request block.

ADDR2 = The address of the buffer address parameter within the second request block.

NOTLOW = A dummy value. If specified, the programmer must provide the core area to be partitioned and its size in words in registers R4 and R3, respectively. If this parameter is left blank, the values "FRCOR$" and "FRLEN$" are loaded into R4 and R3, respectively.

SIZ and BUF= As defined for the EQUATE macro below.

2.7.2 Move a Specified Number of Bytes

This macro generates code to move a specified number of bytes from one location to another.

```
BYTMOV  FROM, REG1, TO, REG2, CNT, REG3
```

which expands to

```
.ENABL  LSB
MOV     #FROM, REG1
MOV     #TO, REG2
```

```
     MOV      #CNT, REG3
1$:  MOVB     (REG1)+,(REG2)+
     SOB      REG3,1$
     .DSABL   LSB
```

where

FROM = Address of first byte to be moved

TO = Address to which the first byte is to be moved.

CNT = The number of bytes to be moved.

REG1, REG2 and REG3 = General Registers.

2.7.3 Definition of System Parameters

This macro defines the system parameters via equate statements.

```
     EQUATE    (no parameters)
```

which expands to (all numbers are octal except where a decimal point appears)

```
CRE  =   0
RET  =   1
RD   =   FO.RD
WR   =   FO.WRT
MO   =   FO.MFY
CONTIG=  1
LINKD=   -1
FLEN =   0
VLEN =   1
CR   =   15
LF   =   12
CHR  =   140
CLN  =   CHR+2
LEN  =   CLN+2
CNT  =   LEN+2
ALC  =   CNT+2
CON  =   ALC+2
REC  =   CON+2
STS  =   REC+2
SIZ  =   STS+4
BUF  =   SIZ+2
```

```
     FHD    =    BUF+2
     FST    =    FHD+2
     LST    =    FST+2
     DFN    =    LST+2
     FNB    =    F.FNB
     NAM    =    FNB+N.FNAM
     EXT    =    FNB+N.FTYP
     DEV    =    FNB+N.DVNM
     UNT    =    FNB+N.UNIT
     LUNS   =    5
     LINMAX=   33
     HDRSIZ=   62
```

where the following pertains to specifying a request block:

```
     CRE    =    Specifies the create function
     RET    =    Specifies the retrieve function
     RD     =    Specifies read only access
     WT     =    Specifies write only access
     MO     =    Specifies read and write access
   CONTIG   =    Specifies a contiguous file
    LINKD   =    Specifies a linked file
     FLEN   =    Specifies a fixed length record
     VLEN   =    Specifies a variable length record
```

The following define ASCII codes:

```
     CR     =    ASCII carriage return
     LF     =    ASCII line feed
```

The following positions of the parameters within the request block relative to the beginning of the block:

```
     CHR    =    Prompt Address
     CLN    =    Prompt Length Address
     LEN    =    Record Length
     CNT    =    Number of Records
     ALC    =    Allocation Flag
     CON    =    Contiguous Records Requested
     REC    =    Record Number Requested
     STS    =    I/O Status Block
     SIZ    =    Buffer Size
```

BUF = Buffer Address

FHD = Blocks in Header

FST = First Record in Buffer

LST = Last Record in Buffer

DFN = Default File Name Block

FNB = File Name Block

NAM = File Name

EXT = File Type

DEV = Device Name

UNT = Unit Number

The following refers to frame displays:

LINMAX = Number of lines of dialogue to allow before rebuilding the display.

The following refers to the header established by the file system within the buffer specified in a request block:

HDRSIZ = Size of file I/O buffer header in bytes.

2.7.4 Definition of Floating Point Registers

This macro defines six mnemonics for use in referencing the floating point processor registers.

```
REGS      (no parameter list)
```

which expands to

```
AC0 = %0
AC1 = %1
AC2 = %2
AC3 = %3
AC4 = %4
AC5 = %5
```

where

AC0 through AC5 = Floating point register mnemonics.

2.7.5 Inserting File Header Text

This macro is used to insert a character string into a file header. The memory address of the beginning of the header is expected to be in R0. Up to three separate character strings can be specified for transfer into the header.

```
    HDRTXT    ADR1, ADR2, ADR3
```

which expands to

```
    ADD       #384., R0
    MOVB      ADR1, R1
    MOVB      (r1)+, (R0)+
    BNE       .-2
    DEC       R0
    MOV       ADR2, R1
    MOVB      (R1)+, (R0)+        This code is generated
    BNE       .-2                 only if ADR2 is not blank
    DEC       R0
    MOV       ADR3, R1
    MOVB      (R1)+, (R0)+        This code is generated
    BNE       .-2                 only if ADR3 is not blank
```

Note that any addressing mode is legal for ADR1, ADR2 and ADR3. For example, if the label "MSO" is attached to a character string its address is specified by "#MSO".

2.7.6 Pushing and Popping Stack Items

These two macros simply push or pop up to six items to or from the stack. If any parameter is omitted, its corresponding line of code is not generated.

To push items onto stack:

```
    PUSH      A, B, C, D, E, F
```

which expands to

```
    MOV       A, -(SP)
    MOV       B, -(SP)
    MOV       C, -(SP)
    MOV       D, -(SP)
    MOV       E, -(SP)
    MOV       F, -(SP)
```

To pop items from stack:

```
    POP       A, B, C, D, E, F
```

which expands to

```
MOV     (SP)+, A
MOV     (SP)+, B
MOV     (SP)+, C
MOV     (SP)+, D
MOV     (SP)+, E
MOV     (SP)+, F
```

2.8 ADDING NEW OPTIONS

Adding a new option to any frame of the IPS software is straightforward if the user is familiar with RSX-11M. There are three steps:

1. Write the program module for the new option. All the subroutines described in this manual are available to the user. The module should be written as a subroutine which expects the following call:

   ```
   JSR     PC, NAME
   ```

   Where name represents the name of the option. The option is to be assembled with the following command:

   ```
   MAC     NAME=MACS, NAME
   ```

   Then loaded into the IPS library:

   ```
   LBR     IPS/IN=NAME
   ```

2. The FRMXX.MAC module, where XX is the desired frame number, is to be updated. The option entry point and option label are to be added. The FRMXX.MAC module is then assembled:

   ```
   MAC     FRMXX=MACS, COMMON, FRMXX
   ```

   Then loaded into the IPS library:

   ```
   LBR     IPS/RP=FRMXX/-EP
   ```

3. The frame overlay descriptor, FRMXX.ODL, must be modified to contain the new option module name. This is usually added as a new branch to the overlay tree. The frame may now be task built as follows:

   ```
   TKB     @FRMXX
   ```

   The frame with the new option is now complete.

(NOTE: In the above procedure, all needed files are assumed to be on the system default device. If this is not the case, add the appropriate device names to the file names.)

SECTION 3

FORTRAN 4-PLUS INTERFACE

Included in the IPS software is a library of Fortran 4-Plus subroutines which interface a user program to the IPS files. The Library is called IOIPS.OLB. It is to be attached to a user program at task build time as follows

```
TKB  TASK = OBJECT, DKN:IOIPS/LB
```

where DKN: is the device the Library resides on. The Task Builder option, "MAXBUF," must be set to 512 bytes.

The Library contains the following subroutines:

RETIPS - Retrieves an existing IPS file
CREIPS - Creates a new IPS file
CLSIPS - Closes an IPS file.
RDUIPS - Reads from an IPS file (unformatted)
WRUIPS - Writes to an IPS file (unformatted)
RDFIPS - Reads from an IPS file (formatted)
WRFIPS - Writes to an IPS file (formatted)
FILNAM - Inputs a file name from the user terminal
HDTXT - Inserts header text into the header
IPSEX - Exits from Fortran Task and alloys entry into another IPS frame
CLEAR - Clears the Tektronix screen

A detailed description of each subroutine follows.

Name: CLEAR

CALL CLEAR

Purpose: Clears the Tektronix screen and sets the cursor to the home position

Arguments:

NONE

Name: CREIPS

```
CALL      CREIPS (LFN, NAME, REC, LEN, HEAD, ERR)
```

Purpose: Create IPS file

Description:

"CREIPS" creates a file called "NAME" with the number of records and record length as specified by "REC" and "LEN". "HEAD" is written into the header block. The logical file number "LFN" is associated with the file. An error code is returned in "ERR". Once a file is created, read and write operations may be performed.

Arguments:

LFN	=	Logical file number (integer). This number is identical to the logical unit number and must be unique. It must not be assigned to any other file.
NAME	=	File name descriptor as an ASCII string terminated by null.
REC	=	Number of records in the file (integer).
LEN	=	Number of bytes per record (integer)
HEAD	=	Header block of file (512 byte array)
ERR	=	Error return code: (integer)
		0 = No Error
		-1 = Error

Example:

```
Call      CREIPS (1, 'DK1.TEST.IMG', 10, 10, IHEAD, IERR).
```

The file 'DK1:TEST.IMG' is created as 10 records, 10 bytes each. The contents of the array IHEAD is written as the IPS header block. No data is actually written into the records. Only space is reserved for them.

Name: CLSIPS

```
CALL      CLSIPS (LFN, DSP, ERR)
```

Purpose: Close IPS file

Description:

"CLSIPS" terminates I/O processing of the file associated to logical file number "LFN". All pending data is written to the file. No further reads or writes may be performed until the file is retrieved by "RETIPS". The file may be deleted by specifying the appropriate "DSP" option. An error code is returned in "ERR".

Arguments:

```
LFN  =    Logical file number of file to close (integer).
DSP  =    Disposition of the file (integer)
               0 indicates to save the file
              -1 indicates to delete the file
ERR  =    Error return code (integer)
               0 = No Error
              -1 = Error
```

Example:

```
CALL      CLSIPS (1, 0, IERR)
FILE  #1 will be closed and saved.
```

Name: FILNAM

```
CALL      FILNAM (PROMPT, NAME, DEVDEF, EXTDEF)
```

Purpose: Enters a filename from the keyboard and inserts defaults.

Description:

"FILNAM" outputs a prompt to the terminal and inputs a character string which is interpreted as a filename. If device or extension are omitted, default values will be inserted into the filename.

Arguments:

PROMPT	=	ASCII string to be output as a prompting message
NAME	=	Array where edited filename is returned as ASCII string terminated by a null (must be dimensioned as 40 bytes minimum)
DEVDEF	=	ASCII string identifying default device (must be followed by a colon)
EXTDEF	=	ASCII string identifying default extension (must be preceded by a period).

Example:

```
CALL      FILNAM ('ENTER FILENAME', NAME, 'DK1:', '.IMG')
```

This call will issue the prompt 'ENTER FILENAME' to the terminal. The user then types in the filename. This routine will check the name for validity, insert defaults if necessary, and return the name in the array "NAME".

Name: HDTXT

```
CALL      HDTXT (TEXT, HEAD)
```

Purpose: Inserts text into header array

Arguments:

TEXT = Character string terminated by a null
HEAD = 512 byte header array

Example:

```
CALL      HDTXT ('TEST IMAGE FILE', HEAD)
```

This call inserts the text 'TEST IMAGE FILE' into the header array "HEAD".

Name: IPSEX

Purpose: Exit from Fortran frame to IPS frames

Description:

"IPSEX" allows a Fortran program to return to an IPS frame, thereby providing continuity to the system. The master option list is displayed and the operator may enter any desired frame number.

Arguments:

NONE

Name: RDFIPS

```
CALL      RDFIPS (LFN, DATA, SIZE, REQ, CON, ERR)
```

Purpose: Read records from IPS file and format

Description:

"RDFIPS" inputs data from an IPS file identified by logical file number "LFN". The data input begins with record "REQ" and ends with "REQ+CON-1". The read is formatted since each byte of the input record is copied to a word in the array "DATA". This is convenient for image files because each pixel may be referenced as a Fortran integer variable. An error code is returned in "ERR".

Arguments:

```
LFN  =   Logical file number of file to read (integer)
DATA =   Array containing data records (integer)
SIZE =   Number of words in array "DATA" (integer)
REQ  =   Requested record number (integer)
CON  =   Number of contiguous records to transfer (integer)
ERR  =   Error return code (integer)
              0 = No Error
             -1 = Error
             -2 = End of File
             -3 = Array too small for requested record
```

Example:

```
CALL      RDFIPS (2,IDATA, 30, 5, 3, IERR)
```

Record numbers 5, 6, and 7 are read into the array "IDATA". Each byte of the records is mapped into the lower byte of each word of the array "IDATA". Since 3 contiguous records are read, each record must be less than or equal to 10 bytes long.

Name: RDUIPS

```
CALL      RDUIPS (LFN, DATA, SIZE, REQ, CON, ERR)
```

Purpose: Read records from IPS file (no formatting)

Description:

RDUIPS inputs data from an IPS file identified by logical file number "LFN". The data input begins with record "REQ" and ends with "REQ+CON-1". The read is a byte for byte copy from the IPS file to the array "DATA". An error code is returned in "ERR".

Arguments:

```
LFN  =    Logical file number of file to read (integer)
DATA =    Array containing data records (byte)
SIZE =    Number of bytes in array "DATA" (integer)
REQ  =    Requested record number (integer)
CON  =    Number of contiguous records to transfer (integer)
ERR  =    Error return code (integer)
               0 = No Error
              -1 = Error
              -2 = End of File
              -3 = Array too small for requested records
```

Example:

```
CALL      RDUIPS (1, DATA, 10, 3, 1, IERR)
```

Record number 3 is read into the array "DATA" from LFN #1. The size of "DATA" is indicated as 10 bytes. Therefore, the record size must be less than or equal to 10 bytes.

Name: RETIPS

```
CALL      RETIPS (LFN, NAME, REC, LEN, HEAD, ERR)
```

Purpose: Retrieve IPS file

Description:

"RETIPS" retrieves an existing file called "NAME", assocites the file with the logical file number "LFN", and prepares the file for I/O. The IPS header block is returned in "HEAD". The number of records and record length are returned in "REC" and "LEN". An error code is returned in "ERR". Once a file is retrieved, read and write operations may be performed. Only an existing and closed file may be retrieved.

Arguments:

LFN	=	Logical File Number (integer). This number is identical to the logical unit number and must be unique. It must not be assigned to any other file.
NAME	=	File name descriptor as an ASCII string terminated by a null.
REC	=	Number of records in the file (integer).
LEN	=	Number of bytes per record (integer).
HEAD	=	Header block of file (512 byte array).
ERR	=	Error return code (integer)
		0 = No error
		-1 = Error

Example:

```
Call      RETIPS (1, 'DK2:GRAY.IMG', IREC, ILEN, IHEAD, IERR)
```

The file "DK2:GRAY.IMG' is opened and may be referenced as LFN #1. Appropriate data is extracted from the IPS file and returned in IREC, ILEN, IHEAD, and IERR.

Name: WRFIPS

```
CALL      WRFIPS (LFN, DATA, SIZE, REQ, CON, ERR)
```

Purpose: Write formatted records to IPS file

Description:

"WRFIPS" outputs data to an IPS file identified by logical file number "LFN". The data output begins with record "REQ" and ends with "REQ+CON-1". The write is formatted since each word of the array "DATA" is copied to a byte in the output file. This is convenient for image files because each pixel may be referenced as a Fortran integer variable. An error code is returned in "ERR".

Arguments:

```
LFN  =    Logical file number of file to write (integer)
DATA =    Array containing data records (integer)
SIZE =    Number of words in array "DATA" (integer)
REQ  =    Requested record number (integer)
CON  =    Number of contiguous records to transfer (integer)
ERR  =    Error return code (integer)
               0 = No Error
              -1 = Error
              -2 = End of File
              -3 = Array too small for requested records
```

Example:

```
CALL      WRFIPS (1, IDATA, 20, 10, 4, IERR)
```

This call copies 4 records, beginning with record 10, to the file identified by LFN #1. Each record must be less than or equal to 5 words each since "IDATA" is dimensioned to be 20 words long.

Name: WRUIPS

```
CALL      WRUIPS (LFN, DATA, SIZE, REQ, CON, ERR)
```

Purpose: Write records to IPS file (no formatting)

Description:

"WRUIPS" outputs data to an IPS file identified by the logical file number "LFN". The data output begins with record "REQ" and ends with "REQ+CON-1". The write is a byte for byte copy from the array "DATA" to the IPS file record. An error code is returned in "ERR".

Arguments:

```
LFN  =    Logical file number of file to read (integer)
DATA =    Array containing data records (byte)
SIZE =    Number of bytes in array "DATA" (integer)
REQ  =    Requested record number (integer)
CON  =    Number of contiguous records to transfer (integer)
ERR  =    Error return code (integer)
               0 = No Error
              -1 = Error
              -2 = End of File
              -3 = Array too small for requested records
```

Example:

```
CALL      WRUIPS (1, DATA, 26, 13, 1, IERR)
```

The contents of byte array "DATA" are copied to the file identified by LFN #1. It is a byte for byte copy to record #13.

SECTION 4

PROGRAM DESCRIPTIONS

A description of each program in the system is contained on the following pages. These programs are one of two types. The first type is designed to be called by the executive (CTLOOP) and therefore the calling sequence is listed as one or more entry points. These routines return control by executing an "RTS PC" instruction.

The second program type consists of one or more callable subroutines. The subroutine calling sequences are detailed complete with calling parameters and descriptions sufficient for the programmer to call upon their services.

Program Name: ARSLCT

Purpose:

To create an image file from a specified rectangular area within an existing image file.

Description:

This routine creates an image from a user-specified area within an existing image. Specification of the area is obtained through the keyboard via the subroutine "SNGDEC". All of the available free core buffer area is used for reading the existing image, extracting the specified area and writing the new image onto the disk.

Subroutines Called:

ERREC, F.C, F.P, F.R, F.S, PARCOR, RAD2AS, SI2DA and SNGDEC

Calling Sequence:

Entry is from the Executive at:

ARSLCT

Program Name: AS2RAD

Purpose:

To pack an ASCII character string into RAD50.

Description:

An ASCII character string is packed into RAD50. Upon detection of an illegal character, the error flag 'BKIND$' is set, and the subroutine returns to the calling program.

Subroutines Called:

RSTREG and SAVREG

Calling Sequence:

```
     MOV            #1$, R5
     JSR            PC, AS2RAD
1$:  BR             .+10.
     .WORD          Address of Executive Common
     .WORD          Address for ASCII string
     .WORD          Address for output RAD50 string
     .WORD          Address of number of ASCII characters
```

General registers are not altered.

Program Name: BLDISP

Purpose:

To display the frame task option menu on the Tektronix display.

Description:

The option menu pointer is retrieved from Common. The screen is cleared, and the cursor is positioned at the top center of the display. The time of day is read and the current tick count is used as a random number to jitter the starting point of the menu to prolong screen life. The option list is then output on successive lines.

Subroutines Called:

ALPHA, CLEAR, GRMODE, HOME, PLOT, RSTREG, SAVREG and TTYOUT

Calling Sequence

```
JSR          PC, BLDISP
```

R1 must contain the address of the Executive Common

Program Name: CB2DA

Purpose:

To convert single or double word binary numbers to decimal ASCII.

Description:

Two modes of parameter passing are offered for complete flexibility. They can be passed in general registers or in parameter lists. For efficiency this routine was programmed to use the floating point processor. The MODF instruction is extremely convenient, especially for double precision numbers. The number of non-blank characters converted is returned in common variable BKIND$.

Subroutines Called:

RSTFPS, RSTREG, SAVFPS and RSTREG

Calling Sequence:

```
     Convert double word to ASCII:
     JSR        PC, DI2DAR
                R3 must contain high order part of number
                R4 must contain low order part of number
                R2 must contain the address of a twelve-byte buffer
                   for the output string
                R1 must contain the address of Executive Common
or,
     MOV        #1$, R5
     JSR        PC, DI2DA
1$:  BR         .+8
     .WORD      Address of double word number (low order part first)
     .WORD      Address of a twelve-byte return buffer
     .WORD      Address of Executive Common

     Convert single word to ASCII
     JSR        PC, SI2DAR
                R4 must contain the single word integer
                R2 must contain the address of a twelve-byte buffer
                   for the output string.
                R1 must contain the address of Executive Common
```

Program Name: CB2DA (Continued)

or,

```
     MOV            #1$, R5
     JSR            PC, SI2DA
1$:  BR             .+8.
     .WORD          Address of a single word number
     .WORD          Address of a twelve-byte return buffer
     .WORD          Address of Executive Common
```

General registers are not altered.

Program Name: CHCUR

Purpose:

To capture the position of the Tektronix cross hair cursor.

Description:

This routine turns on the Tektronix cross hair cursor and waits for the user to select a position. After a position is selected and a character is struck this program accepts the input byte stream. This byte stream is then converted to the coordinate values which are placed in the parameter list. The character struck by the user is also returned.

Subroutines Called:

ALPHA, TTGRIN, RSTREG and SAVREG

Calling Sequence:

```
     MOV          #1$,R5
     JSR          PC, CHCUR
1$:  BR           .+12
     .WORD        Address of Executive Common
     .WORD        Address of character struck on keyboard
     .WORD        Address of X coordinate value
     .WORD        Address of Y coordinate value.
```

General registers are not altered.

Program Name: CLCV

Purpose:

To remove all points from a binary image that are not part of a closed curve.

Description:

First the input file is copies to the output file. All further operations are then performed on the output file. The neighborhood of each pixel of value 255 is examined to determine if it is a part of a closed curve. If only one or no neighbors are 255, then the center pixel is changed to zero. If more than one neighbor is 255, then the surround is checked for an "edge, no edge, edge, no edge" condition. If the condition exists, the point is not changed. If it does not exist, the point is changed to zero.

Subroutines Called:

ERFAT, F.C, F.P, F.R, F.S and RAD2AS

Calling Sequence:

Entry is from the Executive at:

CLCV

Program Name: CMPRES

Purpose:

To compress and expand matrices.

Description:

This program contains two subroutines. The first removes row and column elements of either a matrix or a vector according to a calling parameter list. The second subroutine inserts zeroes into row and column elements of either a matrix or vector, also according to a calling parameter list.

Subroutines Called:

RSTREG and SAVREG

Calling Sequence:

Expand a matric or vector:

```
     MOV     #1$, R5
     JSR     PC, EXPAND
1$:  BR      .+10.
     .WORD   Address of address of list of row and column elements
             at which to insert zeroes
     .WORD   Address of address of matrix or vector
     .WORD   Address of number of rows in matrix
     .WORD   Address of number of columns in matrix.
```

Compress a matrix or vector:

```
     MOV     #1$, R5
     JSR     PC, CMPRES
1$:  BR      .+10.
     .WORD   Address of address of list of row and column elements
             to remove
     .WORD   Address of address of matrix or vector
     .WORD   Address of number of rows in matrix
     .WORD   Address of number of columns in matrix.
```

General registers are not altered.

Program Name: COMBIM

Purpose:

To compute the average, a scaled difference or the absolute difference of two images.

Description:

Two input images are combined point by point according to the entry point selected. The average is formed by adding the points and dividing by two. The scaled difference is the difference between the points plus 255, all of which is divided by two. The absolute difference is simply the absolute value of the difference between the points.

Subroutines Called:

ERFAT, F.C, F.P, F.R, F.S, PARCOR and RAD2AS

Calling Sequence:

Entry is from the Executive at:

AVE - Average images

SCADIF - Scaled difference

ABSDIF - Absolute difference

Program Name: COMPIL

Purpose:

To compile logical and arithmetic statements into executable machine code.

Description:

Two entry points to this routine are available. One accepts a logical expression from the keyboard and compiles it into machine code. This was written for the Boolean logic option. The second entry point allows several complete statements to be entered. Three statement types are recognized. The first is simply an arithmetic assignment statement, the second is like a FORTRAN logical IF statement and the third is a modification of the IF called an IFGO. This is simply a logical IF that when satisfied, the arithmetic assignment is evaluated and all following statements are skipped.

Both entry points return the compiled code in the same format: the compiled code, a list of symbols used in the statements, the location within the code of the values represented by the symbols and the ASCII string consisting of the user entered statements.

Subroutines Called:

AS2RAD, ERFAT, ERREC, RSTFPS, RSTREG, SAVFPS, SAVREG, TTYIN and TTYOUT

Calling Sequence:

To compile assignments, IF and IFGO statements:

```
         MOV      #1$, R5
         JSR      PC, COMPIL
1$:      BR       .+6
         .WORD    Address of a working buffer address
         .WORD    Address of size of buffer in words
```

To compile a logical expression only:

```
         MOV      #1$, R5
         JSR      PC, CMPILB
1$:      BR       .+6
         .WORD    Address of a working buffer address
         .WORD    Address of size of buffer in words
```

The buffer supplied to the two subroutines should be made as large as possible to prevent overflow. Upon return the buffer contains the following:

Program Name: COMPIL (Continued)

Word	Definition
1	Number of words used in buffer.
2	Address of compiled code relative to word 1.
3	Address of symbol table relative to word 1.
4	Address for symbol values relative to word 1.
5+	Input ASCII character string terminated with a null.
	.
	.
	.
	Symbol Table
	.
	.
	.
	Compiled Code
	.
	.
	.
	Symbol Values

SYMBOL TABLE FORMAT

Word	DEFINITION
1	Number of symbols
2	Status word for symbol 1
3,4	RAD50 pack of symbol 1
5	Status word for symbol 2
6,7	RAD50 pack of symbol 2
	.
	.
	.
	Status word for symbol N
	RAD50 pack of symbol N

Program Name: COMPIL (Continued)

STATUS WORD FORMAT

Bit 0	0 = Symbol is not used as an independent variable
	1 = Symbol is used as an independent variable
Bit 1	0 = Symbol is not used as a dependent variable
	1 = Symbol is used as a dependent variable
Bit 2	0 = Non-neighborhood symbol
	1 = Neighborhood symbol

The compiled code is called as follows:

```
JSR     PC,(adr. of code)
```

Prior to this call, the values of all variables used only as dependent variables must be defined. This values are placed in the buffer area identified by word 4 of the return buffer. The first four words of this area are reserved for special subroutine addresses. The first three should be filled with the entry point addresses of the EXP, LN and LOG routines respectively. The fourth is a special routine that retrieves image points whose positions are relative to the current point. This routine must be provided fi the code that is entered references it. The routine is called as follows:

```
JSR     R5,(adr. of neigh. ref. routine)
.WORD   Image number
.WORD   Row offset from current position
.WORD   Column offset from current position
```

(NOTE: This capability is not currently used by the system. It is planned for use in future work)

Following the four special addresses appear the symbol values in the order of appearance within the symbol table. All values are in two word floating point. Independent variable values will be set upon return.

When "CMPILB" is called the logical result is returned in R1. The returned value is either a 1 for true or a 0 for false.

Program Name: CREFLT

Purpose:

To generate a filter file.

Description:

This routine allows a filter to be generated as a function of the row and column position within the filter array or as a function of the Euclidean distance from the upper left corner of the array. The function is defined by the user at the keyboard. The routine "COMPIL" is called to accept this input and to compile the function into machine code.

Prior to generating the filter file, user specified cross sections of the specified filter function are displayed on the graphics display. These cross sections must be perpendicular to the X-Y plane.

In addition to allowing a filter file to be generated, the specified function can be stored in a separate file. This allows it to be recalled in the future for further filter generation.

Subroutines Called:

BLDISP, COMPIL, ENABLR, EXP, ERFAT, ERREC, FLTENT, FPSQRT, F.CL, F.C, F.P, F.R, GETCON, LN, LOG, RAD2AS, SI2DA, SNGDEC, TTYIN, and TTYOUT

Calling Sequence:

Enter is from the Executive at:

CREFLT

Program Name: CRESFS

Purpose:

To create and edit spectral and vector set files.

Description:

This routine performs one of two functions based upon the entry point selected. One entry point deals with spectral sets and the other with vector sets. Similar operations are performed by both routines. File names are entered at the keyboard which are then stored in a file. For spectral file sets, class symbols and data reduction factors are also accepted and stored.

A second operating mode can be entered in which the file name lists can be edited and/or displayed.

Subroutines Called:

BLDISP, CLEAR, DSABLR, ENABLR, ERREC, F.CL, F.C, F.P, F.R, F.S, GETNM, LNPRNT, LPCLOS, RAD2AS, RSTREG, SAVREG, SI2DA, SI2DAR, SNGDEC, TTYIN, TTYGRF, and TTYOUT.

Calling Sequence:

Entry is from the Executive at:

CRESFS - spectral set file operations
CRESET - vector set file operations

Program Name: CREVEC

Purpose:

To create vector files and to add measurements to existing vector files.

Description:

This routine creates two types of vector files. A design vector file is created from a spectral set in which regions have been described. In this case each vector can be identified as belonging to a particular class. A test vector file is created from all points within the spectral set. Since the vectors are not created on the basis of regions, an identifying class symbol cannot be assigned.

Only spatial measurements can be added to an existing vector file. These measurements consist of basic statistics computed over neighborhoods of each point.

Subroutines Called:

BEGFND, CFREQ, CMEAN, CSDEV, DI2DAR, ERREC, ERFAT, F.CL, F.P, F.R, F.S, FINDPT, GHIGH, GLOW, GMEAN, GMEDN, GRANGE, GSDEV, RAD2AS, SI2DA, SMINIT, SNGDEC, SNGLPT, TTYIN, TTYGRF and TTYOUT

Calling Sequence:

Entry is from the Executive at:

CREVEC - Create test vector file
CREDES - Create design vector file

Program Name: CRLOG

Purpose:

To create and initialize logic tree files.

Description:

This routine creates a logic tree file. The directory, the logic tree and the class symbol pages are all initialized according to the current vector set. The overall result is a logic tree consisting of only the senior node. All class symbols are assigned to that node.

Subroutines Called:

ERFAT, F.C, F.CL, F.P, RSTREG and SAVREG

Calling Sequence;

```
     MOV      #1$, R5
     JSR      PC, CRLOG
1$:  BR       .+8.
     .WORD    Address of the address of a working buffer
     .WORD    Address of the size of the buffer
     .WORD    Address of a four word file specification
              (Device (RAD50), unit and file name (RAD50))
```

Program Name: CTFILE

Purpose:

To create an input file from values entered via the keyboard.

Description:

The values for each pixel within the image are obtained through the keyboard. The values are then stored in a disk file.

Subroutines Called:

F.CL, F.P, F.S, SI2DA, SNGDEC, and TTYOUT

Calling Sequence:

Entry is from the Executive at:

CTFILE

Program Name: CTLOOP

Purpose:

To control execution of each Frame Task.

Description:

This routine is the main control loop for each Frame Task. It is entered from the task restart routine. The routine processes all option and frame selections, and also allows insertion of comments into the log file. An exit route to the RSX-11M Executive is also provided.

Subroutines Called:

CLEAR, ERREC, LOGIT, STRTSK, TTYIN, TTYOUT

Calling Sequence:

The routine is entered from the task restart routine following a task start or fatal error by:

```
JMP             CTLOOP
```

R4 must contain the address of the Executive Common

It is reentered from option processing subroutines by executing an

```
RTS             PC
```

Option subroutines need not save the general registers.

Program Name: DECIM

Purpose:

To reduce an image in size by discarding pixels.

Description:

The image is reduced by retaining every Nth row and every Mth column. The result is stored in a disk file.

Subroutines Called:

ERREC, F.C, F.P, F.R, F.S, PARCOR, RAD2AS, SI2DA and SNGDEC

Calling Sequence:

Entry is from the Executive at:

    DECIM

Program Name: DECIMG

Purpose:

To create a thematic map from a classified set of vectors.

Description:

A thematic map is created to present the classification of vectors in an image format. The first step in the process is to insure that each terminal node in the tree is associated with only one class. Also, the vector file must contain positional information with respect to rows and columns. The classification of each vector is determined and a grey level is assigned to the pixel value associated with the position of the vector. Grey level assignments are made by the user on the basis of class.

Subroutines Called:

CLLF, DI2DAR, ERFAT, ERREC, F.C$, F.P, F.R, F.R$, F.SH, GETNAM, GETNOD, GETSYM, LNPRNT, LPCLOS, OPLF, RAD2AS, SI2DAR and SNGDEC

Calling Sequence:

Entry is from the Executive at:

DECIMG

Program Name: DELETE

Purpose:

To delete a file from a disk peripheral.

Description:

The file specification is obtained from the user. The file is then deleted using the FCS Delet$ Macro. A loop is contained in the program to allow several names to be entered. For successive names the previous file name extension is used if none is entered.

Subroutines Called:

ERFAT, ERREC and GETNAM

Calling Sequence:

Entry is from the Exeuctive at:

```
DELETE
```

Program Name: DENHST

Purpose:

To compute and display histograms of images and regions within images.

Description:

The histograms computed are based upon the density values within the image and the maximum, minimum and average difference between each point and its eight neighbors. These are computed for complete images and for specified regions within images. The routine "FINDPT" is used to determine which points are within the regions. After the histogram is computed, the routine "HISPLT" is called to display it on the graphics display.

After the histogram is displayed several options are available. Most of these deal with such things as zooms, change tick mark spacing, etc. One option determines the percentage of the total number of points that lie within a user specified range. Another allows a cumulative histogram to be displayed.

In addition to above options, a density profile of any given row can be displayed in histogram format. Display options similar to those discussed above are also available for this function.

Subroutines Called:

ALPHA, BEGFND, BLDISP, CLEAR, DBLDEC, DI2DA, DI2DAR, ERFAT, ERREC, FPASCF, F.CL, F.P, F.R, FINDPT, GRMODE, HISPLT, HOME, OPTION, PLOT, RAD2AS, RSTREG, SAVREG, SI2DA, SI2DAR, SNGDEC, TTYGRF and TTYOUT

Calling Sequence:

Entry is from the Executive at:

IMHIS - Calculate image histograms

REGHIS - Calculate region histograms

LINPRF - Display single line density profile

Program Name: DICOMD

Purpose: To record images on film via the DICOMED Image Recorder.

Description:

The user is prompted to enter recorder resolution, Polarity transfer function and blow up factor. Next, the recorder is initialized and the output loop is entered. Each line is read from the disk file and prepared for output. The line is then printed the number of times equal to the blow-up factor. When the operation is complete the user is informed, the file is closed, and control returns to the executive.

Subroutines Called:

ERFAT, ERREC, F.P, F.R, F.S, RSTREG, SAVREG, SNGDEC, TTYOUT

Calling Sequence:

Entry is from the executive at:
DICOMD

Program Name DIFFAC

Purpose:

To create a scaled, weighted combination of two images.

Description:

Each point in the two input images is multiplied by a factor. The combination is then formed according to the signs of the weights. The result is then scaled over the range of zero to 255.

Subroutines Called:

ERFAT, ERREC, F.CL,F.P, F.R, F.S, PARCOR, RAD2AS, SI2DA, SI2DAR, SNGDEC and TTYOUT

Calling Sequence:

Entry is from the Executive at:

DIFFAC

Program Name: DIGIN

Purpose: To input DICOMED Digitizer image to disk

Description:

The user is prompted to enter the digitizer resolution. The digitizer is initialized and the input loop entered. As a line is digitized it is transferred to the image file. When the operation is complete the user is informed, the file is closed, and control returns to the executive.

Subroutines Called:

ERREC, F.CL, F.P, F.R, RSTREG, SAVREG, SNGDEC, TTYOUT

Calling Sequence:

Entry is from the executive at:
DIGIN

NOTE: This option is not fully implemented.

Program Name: DIRCTY

Purpose:

To print directories of disks on the line printer or graphics display.

Description:

The directory is searched for all matching files and their file ID's are saved in a temporary work file. Then the system index file is opened and the file creation date is extracted from each file header. Each file is then opened to extract block count, record count, record length and header text. The work file is then sorted alphabetically, first by extension and then by name. The work file is then dumped in either short or long format to the line printer or terminal.

Subroutines Called:

BLDISP, CLEAR, ERFAT, ERREC, GETNAM, LNPRNT, LPCLOS, RAD2AS, SI2DA, SNGDEC, TTYIN and TTYOUT

Calling Sequence:

Entry is from the Executive at:

DIRLNG - List long directory
DIRSRT - List short directory

Program Name: ELCHNG

Purpose:

To replace specified density values or ranges of values with new values.

Description:

The type of operation, individual values or ranges, is determined by the entry point. In either case, the values and ranges along with the replacement values are specified by the user. A table of 256 entries is then created which defines a transfer function over the grey value range of 0 to 255. This table, along with the input and output files, is passed to TRNFTN for processing.

Subroutines Called:

ERREC, F.C, F.P, F.R, F.S, RAD2AS, SNGDEC, TRNFTN and TTYOUT

Calling Sequence

Entry is from the Executive at:

ELCHNG	-	Change individual elements
TRSHLD	-	Change ranges of elements

Program Name: ERMES

Purpose:

To output a selected error message to the terminal.

Description:

This routine searches the file ERRMESFIL.IPS for an error message based on the error number and entry point (either recoverable or fatal). If the request is valid, the message is output to the terminal. Recoverable errors result in a return to the calling program. Fatal errors result in an exit to the executive.

Subroutines Called:

F.S, RSTREG, SAVREG, TTYOUT

Calling Sequence:

To output a recoverable error message:

```
     MOV        #1$, R5
     JSR        PC, ERREC
1$:  BR         .+6
     .WORD      Address of Executive Common
     .WORD      Address of error number
```

To output a fatal error message:

```
     MOV        #1$, R5
     JSR        PC, ERFAT
1$:  BR         .+6
     .WORD      Address of Executive Common
     .WORD      Address of error number
```

General registers are not modified.

Program Name: EVALBL

Purpose:

To apply Boolean logic to a vector set.

Description:

This routine is called by EVALDR to apply Boolean logic which resides at a given node in the logic tree. The logic file is opened, the logic is extracted and the file is closed. The vector set is then opened and the logic is applied to each vector. The vector set is then closed upon completion.

Subroutine Calls:

CLLF, CLOVEC, EXP, GETVEC, GTLF, LN, LOG, OPLF, OPNVEC, RSTFPS, RSTREG, SAVFPS, SAVREG

Calling Sequence:

```
JSR     PC, EVALBL
```

Prior to entry, the first nine words of the "FRECOR" buffer must be set to:

WORD	DESCRIPTION
0	Logic node number
1	Logic file device (RAD50)
2	Logic file unit number
3,4	Logic file name (RAD50)
5	Vector file device (RAD50)
6	Vector file unit number
7,8	Vector file name (RAD50)
9	Node number at which logic evaluation began (used only by EVALDR)

These values are not modified.

Program Name: EVALDR

Purpose:

To control the overall logic evaluation operations.

Description:

This routine accesses the logic file to determine which logic evaluation routine should be called at each node in the tree. That routine is then called.

Three modes of operation exist. The first mode evaluates the logic at the current node only. The second mode evaluates logic at the current node and all nodes below the current node. The last mode evaluates all logic in the tree beginning at the senior node.

Subroutine Calls:

CLLF, CLLOGF, EVALBL, EVALFP, ERFAT, GETNAM, GETNOD, GTLOGF, OPLF and OPLOGF

Calling Sequence:

Entry is at:

EVALDR	-	OVERAL EVALUATION
EVACRE	-	AFTER LOGIC CREATION
EVASEL	-	AFTER SELECTING LOGIC FILE

Program Name: EVALFP

Purpose:

To apply Fisher pairwise logic to a vector set.

Description:

This routine is called by EVALDR to apply Fisher logic which resides at a given node in the logic tree. The logic file is opened, the logic is extracted and the file is closed. The vector set is then opened and the logic is applied to each vector. The vector set is then closed upon completion.

Subroutine Calls:

CLLF, CLOVEC, GETVEC, GTLF, OPLF, OPNVEC

Calling Sequence:

```
JSR        PC, EVALFP
```

Prior to entry, the first nine words of the "FRECOR" buffer must be set to:

WORD	DESCRIPTION
0	Logic node number
1	Logic file device (RAD50)
2	Logic file unit number
3,4	Logic file name (RAD50)
5	Vector file device (RAD50)
6	Vector file unit number
7,8	Vector file name (RAD50)
9	Node number at which logic evaluation began (used only by EVALDR)

These values are not modified.

Program Name: EXP

Purpose:

To compute the exponential value for a given power.

Description:

This routine raises "e" to the power indicated by the calling argument. The computational method is discussed in Hart[1]. Upon under or overflow, the error flag 'BKIND' is set and the argument is unchanged.

Subroutines Called:

SAVFPS, SAVREG, RSTEPS, RSTREG

Calling Sequence:

```
     MOV       #1$, R5
     JSR       PC, EXP
1$:  BR        .+6
     .WORD     Address of Executive Common
     .WORD     Address of single precision FP argument
```

General and floating point registers are unmodified.

Program Name: FDBCRT

Purpose:

Allocate file descriptor blocks for the Error, Log and Master Option files.

Description:

This module reserves and initializes two file descriptor blocks. The first is used by the user Log File. The second is shared by the Error file and the Master Option File. Record deblocking buffers are also allocated. The FDB addresses are global symbols, allowing their values to be stored in the Executive Common.

Program Name: FILE1

Purpose:

To create and retrieve data files

Description:

This program has five entry points. Two entry points exist for creating files and two exist for retrieving files. One entry point in each case allows the file to be created or retrieved without opening it for access. The other entry point opens the file for access as well. A fifth entry point simply checks for the existence of a file. If it is not found, an alternate return is taken.

A list is maintained within EXEC of all open files. Therefore, as each file is opened, the address of its request block is inserted into this list.

All file operations are performed within the RSX-11M file structure. The RSX-11M FCS Macroes are used for accomplishing the desired tasks.

Subroutines Called:

ERFAT, ERREC, GETNAM, RDIT, RESTRT, RSTREG, SAVREG, TTYIN, TTYOUT, WTIT

Calling Sequence:

```
     To create a file
     MOV            #1$, R5
     JSR            PC, F.C$
1$:  BR             .+12.
     .WORD          Address of Executive Common
     .WORD          0
     .WORD          0
     .WORD          Address of File Request Block
     .WORD          Address of Null Input Indicator
```

(FILE1 cont.)

```
To create and open a file
JSR             PC, F.C

The argument list is identical to F.C$

To retrieve a file
JSR             PC, F.R$

The argument list is identical to F.C$

To retrieve and open a file
JSR             PC, F.R
The argument list is identical to F.C$

To check for a file's existance
MOV             #1$, R5
JSR             PC, F.CH
1$: BR          .+12.
    .WORD       Address of Executive Common
    .WORD       0
    .WORD       0
    .WORD       Address of File Request Block
    .WORD       Address of Return Indicator
                   Set to 0 if file is found
                   Set to -1 if file is not found
```

(FILE1 cont.)

The File Request Block includes the following data, which is usually generated by the F.REQ macro.

Location	Description
FRBFDB	Start of RSX-11M File Descriptor Block
FRBCHR	Address of prompt for filename request
FRBCLN	Address of length of prompt
FRBLEN	Record length
FRBCNT	Number of records in file
FRBALC	Linked/Contiguous flag
FRBCON	Number of contiguous records required in memory
FRBREC	Requested record number
FRBSTS	Disc I/O Status Block
FRBSIZ	Buffer size (bytes)
FRBBUF	Buffer address
FRBFHD	Number of device blocks in file header
FRBFST	First record currently in buffer
FRBLST	Last record currently in buffer
FRBDFN	Default file name block for device and file extension

Program Name: FILE2

Purpose:

To access data within a file.

Description:

This routine accesses data within files that are opened by the routines in FILE1 program.

When a request is made for a record or group of contiguous records, the associated buffer header is examined to determine if the data has been loaded into memory by a previous operation. If it is in memory, it's address is simply returned to the calling program. If not, the file access mode is checked to determine what operations are to be performed. If the access is "read only", the requested data is read into memory. If the access is "write only", the data is written onto the disk and the position for the desired record(s) is established in memory. If the access is "read/write", the current buffer contents are written onto the disk and the desired record(s) are then read into memory. In all cases where data is read into memory, an amount of the data equal to the size of the buffer is read into memory. This is done in anticipation of future requests for the next records in sequence. This reduces overall data transfer time.

Subroutines Called:

ERREC, ERFAT, RSTREG, SAVREG

Calling Sequence:

```
     MOV          #1$, R5
     JSR          PC, F.P
1$   BR           .+8
     .WORD        Address of Executive Common
     .WORD        Address of File Request Block
     .WORD        Address of Nonexistant Record Indicator
```

(FILE2 cont)

This program also includes three service routines which are used by FILE1 and FILE2.

To write the data defined by the FDB:

```
     JSR            PC, WTIT
```

R0 must contain the FDB address

To read the data defined by the FDB:

```
     JSR            PC, RDIT
```

R0 must contain the FDB address

To set up the FDB to read or write the data currently in the buffer:

```
     JSR            PC, COMPU
```

R0 must contain the address of the File Request Block.

Program Name: FILE3

Purpose:

To close files.

Description:

Two entry points are available for closing files. The first closes individual files and the second closes all files that are currently open. In either case, the associated file request block addresses are removed from the open file list within the Executive Common. Files with a .TMP extension are deleted.

Subroutines Called:

COMPU, ERFAT, ERREC, RDIT, RSTREG, SAVREG, WTIT

Calling Sequence:

To close a file:

```
     MOV            #1$, R5
     JSR            PC, F.CL
1$:  BR             .+6
     .WORD          Address of Executive Common
     .WORD          Address of File Request Block
```

To close all open files:

```
     MOV            #1$, R5
     JSR            PC, F.S
1$:  BR             .+6
     .WORD          Address of Executive Common
     .WORD          0
```

Program Name: FILE4

Purpose:

To extend the length of a specified file.

Description:

The file is retrieved and its current size is obtained. The requested extension is then added to this size and a new version number is created under the same name. The old file is copied into the new file, and old file is deleted.

Subroutines Called:

ERFAT, ERREC, RSTREG and SAVREG

Calling Sequence:

```
     MOV        #1$, R5
     JSR        PC, F.E
1$:  BR         .+4
     .WORD      Address of File Request Block
     .WORD      Address of desired file extension in disk block.
```

Program Name: FILL

Purpose:

To restore breaks in lines in a binary image.

Description:

This routine scans a binary input image for non-edge points (0 grey value) whose eight surrounding points meet an "edge, non-edge, edge, non-edge" condition. When this condition is discovered the point is replaced with a value of 255. The result is non-connected edge points separated by one pixel are connected. Two different procedures are followed depending upon which entry point is selected. Either the operation is performed on only data appearing in the input image or on the current state of the union of the input and output images.

Subroutines Called:

ERFAT, F.C, F.P, F.R, F.S and RAD2AS

Calling Sequence:

Entry is from the Executive at:

REGFIL - Regular fill in
ADPFIL - Adaptive fill in

Program Name: FINDPT

Purpose:

To determine the interior points of a region defined by a region file.

Description:

This program determines the points interior to and on a region boundary. A boundary, as opposed to being a line of zero width, is actually one or more pixels wide depending upon the orientation of the line within the array. As a result, this "thick" boundary greatly complicates the task of finding the region points.

The problem is attacked by taking one row at a time and determining its intersection with the boundary lines. These are called intersection intervals. The beginning of a row is assumed to be an exterior point and is therefore a point of reference. When the row reaches an intersection interval the points then are considered interior points. After the interval is passed, the status of the points depends on the interval characteristic. If the intersection interval is not at the end of a boundary line segment, then as the row passes through the interval, it goes from outside the region to inside or vice-versa. Therefore the interval is an "in-out" or "change" interval.

The intervals at the vertices of the line segments are somewhat more complicated. In this case, the characteristic of the interval is "change" if the boundary lines meeting at the vertice approach it from opposite sides of the row in question. If they approach the vertice from the same side, the interval is a "no change" interval. This means that points on the interval are within the region but points on either side of it are either both inside or both outside the region.

Horizontal lines in the boundary are essentially one long intersection interval. Such an interval's characteristic is determined by the lines which meet its end points. A rule similar to the vertice rule above is followed. If the lines at each end approach from opposite sides of the row its characteristic is "change". If they approach from the same side it is "no change". Successive horizontal lines are combined into one long interval.

For each row that crosses the region, the above procedure is followed to determine the interior points. One subroutine (BEGFND) must be called to initialize the operation. Successive calls to FINDPT then return the row and column values found within the region. Each call returns one point.

Another subroutine (SNGLPT) determines if a given point is in the region. The procedure for making this determination is the same as the above.

Program Name: FINDPT (Continued)

Subroutines Called:

ERFAT, F.CL, F.P, F.R, RSTFPS, RSTREG, SAVFPS and SAVREG

Calling Sequence:

To retrieve the region file and initialize the working buffer for use by FINDPT and SNGLPT:

```
        MOV     R5,-(SP)
        MOV     #1$,R5
        JSR     PC,BEGFND
1$:     BR      2$
        .WORD   Address of the request block for the region file..
        .WORD   Address of an alternate return to be taken when the
                user responds to the request for a region file name with
                only a carriage return.
2$:     MOV     (SP)+,R5
```

Upon return from BEGFND the buffer specified in the request block is still in use even though the region file has been closed. This buffer is used until all calls to FINDPT and SNGLPT have been completed for that region.

To obtain the coordinate of the next point found within the region:

```
        MOV     R5,-(SP)
        MOV     #1$,R5
        JSR     PC,FINDPT
1$:     BR      2$
        .WORD   Address of the request block for the desired region
        .WORD   Address of an alternate return when there are no further
                points remaining in the region.
2$:     MOV     (SP)+,R5
```

The coordinate values are returned on the stack with the row on top followed by the column.

Program Name: FINDPT (Continued)

To determine if a particular point is within the region:

```
        MOV     #1$,R5
        JSR     PC,SNGLPT
1$:     BR      2$
        .WORD   Address of the request block for the desired region
        .WORD   Row coordinate value
        .WORD   Column coordinate value
        .WORD   In/Out Indicator (0=Out, 1=In)
2$:
```

Program Name: FISODP

Purpose:

To compute the Fisher direction, the discriminant plane vector and five thresholds.

Description:

This routine computes the above mentioned values for a given class pair. The records in the mean-covariance file (MCFILE.MC) associated with the two classes are passed as parameters. One class pair is treated per call.

Subroutines Called:

CMPRES, ERFAT, EXPAND, FPSQRT, F.P, GETIX, INVMAT, LINDEP, MDOTV, RSTREG, SAVREG and SETIXD

Calling Sequence:

Prior to calling "FISHER" the following call must be made:

```
JSR     PC,SETFIS
```

this routine expects the following globals to be provided:

BOTCOR - Address of the beginning of a working buffer

TOPCOR - Address of the end of the working buffer

The Fisher values are calculated by the following call:

```
JSR     PC,FISHER
```

This routine expects the following globals to be provided:

MCBLK - Address of request block for file MCFILE.MC

MCREC - Record number parameter in request block for MCFILE.MC

MVEC - Buffer containing the measurements from which to create the logic. The first word is the number of measurements and each word thereafter contains a measurement number.

PAGEA - Record number in MCFILE.MC corresponding to the first class.

PAGEB - Record number in MCFILE.MC corresponding to the second class.

VECDIM - Dimension of the vectors.

Upon return, the computed values are found at the addresses given in the following global locations:

DP - Address of discriminant plane vector

FISH - Address of Fisher vector

THRESH - Address of five thresholds.

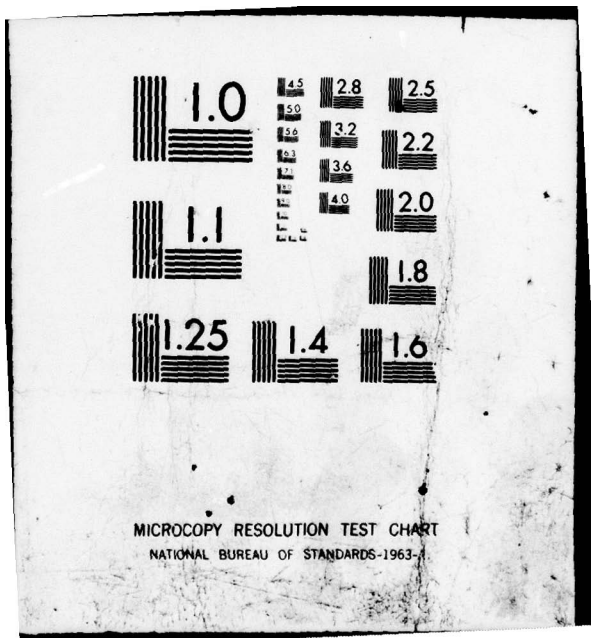
FISODP (Continued)

All values are in single precision floating point. See documentation of logic tree file (Appendix E) for the formats of the return values.

General Registers are not modified.

AD-A073 653 AMHERST SYSTEMS INC BUFFALO NY F/G 9/2
IMAGE PROCESSING SYSTEM SOFTWARE. VOLUME II. PROGRAMMING MANUAL--ETC(U)
JUN 79 E G EBERL, P T GLINSKI F30602-78-C-0077
UNCLASSIFIED AMHERST-0077-VOL-2 RADC-TR-79-52-VOL-2 NL

2 OF 3
AD A073653


1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
4.5
5.0
5.6
MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

Program Name: FLTMUL

Purpose:

To form the product between a filter file and a file of another type.

Description:

The product between the files is formed on a point-by-point basis. The second file can have byte, integer, double integer or floating point values. However, it must be an array type of file such as an image or a Hadamard transform of an image.

The output file is created in the same size and format as the second input file. If at any point overflow occurs, the output takes on the largest positive or negative value whichever is appropriate. A count of overflows is maintained and is reported to the user at the end of processing.

Subroutines Called:

DI2DA, ERFAT, F.C, F.P, F.R, F.S, PARCOR, RAD2AS, RSTFPS, SAVFPS and TTYOUT

Calling Sequence:

Entry is from the Executive at:

FLTMUL

Program Name: FPASC

Purpose:

To convert from binary floating point to an ASCII character string.

Description:

Two types of conversion are provided, both of which follow FORTRAN conventions. The first is similar to the FORTRAN "F" conversion. That is, the binary value is converted to its equivalent decimal number with a decimal point. The field width and the number of places to the right of the decimal point must be specified.

The second type provides the FORTRAN "E" type conversion. That is, the binary value is converted to a decimal number multiplied by a power of ten. The field width and the number of places to the right of the decimal point must be specified in this case also.

Subroutines Called:

SAVFPS, SAVREG, RSTFPS and RSTREG

Calling Sequence:

Fixed Format Conversion:

```
     MOV       #1$, R5
     JSR       PC, FPASCF
1$:  BR        .+14
     .WORD     Address of Executive Common
     .WORD     Address of field width
     .WORD     Address of number of digits to the right of the decimal point
     .WORD     Address of single precision floating point number
     .WORD     Address of the buffer for the converted number
```

The field width minus the number of decimal digits must be greater than 1.

Exponential Format Conversion:

```
     MOV       #1$, R5
     JSR       PC, FPASCE
1$:  BR        .+14
```

(The parameter list is identical to that for FPASCF)

The field width minus the number of decimal digits must be greater than 5.

Program Name: FPW

Purpose:

To control the overall process of creating Fisher pairwise logic.

Description:

This routine monitors the processes required for the creation of Fisher logic. It performs the following functions:

1. Allocate buffer space for the various operations.
2. Retrieve the mean and covariance file.
3. Call "FISHER" to compute the Fisher vectors for each class pair.
4. Insert the resulting logic into the current logic file.

Prior to execution, this routine expects that a logic tree node has been selected for the logic and that the appropriate means and covariances have been computed and stored in the file "MCFILE.MC".

Subroutines Called:

ADDNOD, ADDSYM, CLLOGF, ERFAT, FISHER, F.CL, F.P, F.R, GETMEA, GETNOD, GETSYM, GTLOGF, LOADER, OPENLF, OPLOGF, RSTREG, SAVREG, SETFIS, TTYIN

Calling Sequence:

Entry is at:

FPW

Program Name: FPSQRT

Purpose:

To compute the square root of a floating point number.

Description:

The square root of a single or double precision floating point number is computed. The precision is determined from the status register within the floating point processor. An initial guess at the final value is made based upon the half of the exponent value. This results in only three Newton iterations required for single precision and four iterations required for double precision.

Subroutines Called:

SAVFPS, SAVREG, RSTFPS, and RSTREG

Calling Sequence:

```
     MOV     #1$, R5
     JSR     PC, FPSQRT
1$:  BR      .+4
     .WORD   Address of the single or double precision floating
             point argument
```

The result is returned in the subroutine argument. The general and floating point registers are not changed.

Program Name: FRAMES

Purpose: To generate the file MSTROPTON.LST.

Description:

This program converts the file MSTROPTON.SRC to the IPS compatible file MSTROPTON.LST.

Subroutines Called:

NONE

Calling Sequence:

This routine is executed via the RSX RUN command.

Program Name: FRMXX

Purpose:

To provide the individual frame option lists and dispatch tables

Description:

FRMXX actually represents a set of similar modules, where XX is replaced by the various Frame numbers. A separate file is maintained for each Frame. There are currently modules numbered FRM01 through FRM13, representing the thirteen Frames in the system. The modules consist entirely of data, including the text to be displayed by the BLDISP routine, and a dispatch table for processing the various option overlays.

The option list menu is stored at global label OPMENU. It contains the frame header text pointer and option text pointers in the following format.

```
OPMENU:   .WORD     N         ;Number of Options
          .WORD     HDR       ;Address of Frame ID Text
          .WORD     LHDR      ;Address of Text Length
          .WORD     01        ;Address of Option 1 Text
          .WORD     L01       ;Address of Length of Text
          .WORD     02        ;Address of Option 2 Text
          .WORD     L02       ;Length of Text
          .
          .
          .
          .WORD     ON        ;Address of Option N Text
          .WORD     LON       ;Address of Text Length
```

The option dispatch table is stored at global label OPDISP. It contains the entry points for the options in the order specified in the menu.

(FRMXX cont.)

```
OPDISP:   .WORD     ROUT1          ;Entry for Option 1
          .WORD     ROUT2          ;Entry for Option 2
          .
          .
          .
          .WORD     ROUTN          ;Entry for Option N
```

All data storage for the Executive Common is also allocated and initialized by assembling each module with the COMMON file as a prefix file.

Subroutines Called:

None

Calling Sequence:

The initialization code in the Executive Common is executed by specifying it as the entry point of the program, as follows.

```
     .END      FSTRT
```

No other modules may specify entry points.

Program Name: GETBOL

Purpose:

To create Boolean logic.

Description:

This routine directs the overall creation of Boolean logic. The logic is entered at the keyboard as a logical/arithmetic expression. It is accepted and compiled into machine code by the routine "CMPILB". Following its entry the user is given the option to reject it and make a new entry.

After the logic has been accepted, it is stored in the logic file and two subnodes are added to the current node. The class assignments for each subnode are obtained from the user.

Subroutines Called:

ADDNOD, ADDSYM, CLLOGF, CLOVEC, CMPILB, ERREC, GETNOD, GETSYM, GTLOGF, LOADER, OPLOGF, OPNVEC, OPTION, RAD2AS, SI2DA, TTYIN and TTYOUT

Calling Sequence:

Entry is at:

GETBOL

Program Name: GETCON

Purpose:

To plot cross sections of a filter on the graphics display.

Description:

This routine is an overlay to the "CREFLT" routine which creates filter files. The function is passed to this routine as machine code. If any undefined constants exist in the code, this routine requests them from the user. The user is also queried to obtain the end points of a line in the X-Y filter plane through which a perpendicular plane is inserted to obtain a cross section of the function. The cross section is then displayed on the graphics display. The user is then given the following options:

1. Accept the filter function;
2. Reject the function and go back to "CREFLT" to get another;
3. Display another cross section;
4. Exit to the executive.

Subroutines Called:

ALPHA, CLEAR, CREFLT, DSABLR, ERREC, FLTENT, FPASCE, GRMODE, PLOT, RAD2AS, RSTFPS, SAVFPS, SNGDEC, TTYIN, TTYGRF, TTYOUT and WHICH

Calling Sequence:

```
JSR       R5, GETCON
```

Program Name: GETDAY

Purpose: Access date as an ASCII string

Description:

This routine reads the date through RSX-11, converts it to ASCII, and returns it to the user supplied buffer. The format is:

```
        MO-DY-YR
where:  MO = month
        DY = day
        YR = year
```

Subroutines Called:

RSTREG, SAVREG

Calling Sequence:

```
        MOV         #1$, R5
        JSR         PC, GETDAY
1$:     BR          .+6
        .WORD       Address of Executive Common
        .WORD       Address of 8 byte field where
                    ASCII date returned.
```

General Registers are not modified.

Program Name: GETIM

Purpose:

Access time of day as an ASCII string.

Description:

This routine reads the time through RSX-11, converts it to ASCII, and returns it to the user supplied buffer.

The format is:

```
        HR:MN:SC
where:  HR = Hours relative to midnight
        MN = Minutes
        SC = Seconds
```

Subroutines Called:

RSTREG, SAVREG

Calling Sequence:

```
     MOV        #1$, R5
     JSR        PC, GETIM
1$:  BR         .+6
     .WORD      Address of Executive Common
     .WORD      Address of 8 byte field where ASCII time returned.
```

General registers are not modified.

Program Name: GETIX

Purpose:

To retrieve a specified element of a given matrix.

Description:

This routine obtains a matrix element specified by its row and column position. The matrix may be stored in either single or double precision floating point format. As a result, one of two initialization routines must be called prior to requesting any elements.

Subroutines Called:

None

Calling Sequence:

To initialize for single precision:

```
     JSR       PC, SETIXF
```

To initialize for double precision:

```
     JSR       PC, SETIXD
```

To retrieve a matrix element:

```
     MOV       #1$, R5
     JSR       PC, GETIX
1$:  BR            .+6
     .WORD     Address of a location containing the row number
     .WORD     Address of a location containing the column number.
```

Prior to calling GETIX, the following global parameters must be set:

DIMEX - Number of columns in the matrix

MATEX - Core address of the matrix.

The core address of the element is returned in R3 and the element itself is returned in AC3. All other registers are unaffected.

Program Name: GETMEA

Purpose:

To obtain the measurement numbers from the user for Fisher logic creation.

Description:

This routine requests measurement number input from the user. The numbers are accepted as one line of input where the numbers are assumed to be separated by commas. Ranges of measurements are allowed which must be indicated by the limits of the range separated by a dash.

Subroutines Called:

ERREC, RSTREG, SAVREG, TTYIN

Calling Sequence:

```
     MOV      #1$, R5
     JSR      PC, GETMEA
1$:  BR       .+4
     .WORD    ADR
```

where

ADR = Address of a buffer in which to return the measurements. The first word of this buffer contains the number of measurements. Measurement numbers are sorted in increasing order.

All registers are preserved.

Program Name: GETNM

Purpose:

To input file name information into a buffer. This routine emulates the DOS version of the "GETNAM" subroutine.

Description:

A prompt is issued to the terminal requesting a filename. When the filename is entered, it is parsed and its validity checked. If invalid, a new name is requested. The file name is then placed into the user supplied buffer with the following format:

Relative Address Within Buffer	Length	Item	Format
Byte 7	1 Byte	Unit #	Binary
Byte 8	2 Bytes	Device	RAD50
Byte 14	4 Bytes	Filename	RAD50
Byte 18	2 Bytes	Extension	RAD50

This format is identical to that of the DOS link and filename block.

Subroutines Called:

AS2RAD, ERFAT, ERREC, GETNAM, .PARSE, RSTREG, SAVREG

Calling Sequence:

```
     MOV      #1$, R5
     JSR      PC, GETNM
1$:  BR       .+6
     .WORD    Address of message
     .WORD    Address of 12 word return buffer
```

Program Name: GTFILE

Purpose:

To create a checkerboard test image.

Description:

This routine creates an image file of 512 rows and 512 columns. The first row is created with grey value 0 in the first 32 columns, 1 in the next 32 and up to 15 in the last 32. This row is then copied into the image file 32 times. The next row sequence begins with value 16 in the last 32 columns and increases by one every 32 columns until reaching the beginning of the row. This row is then copied into rows 33 thru 64 of the file. This alternating process continues until all rows are complete.

Subroutines Called:

ERFAT, ERREC, F.CL, F.C and F.P

Calling Sequence:

Entry is from the executive at:

GTFILE

Program Name: HADMAR

Purpose:

To perform a scaled Hadamard transform on a one-dimensional array of double precision integers.

Description:

This routine computes the Hadamard transform in the classical manner of log N steps where N is the number of elements. The sum and difference operations are performed between two buffers. The first buffer is the input array and the second is a working buffer. If N is odd, the result appears in the working buffer. In this case, it is copied back into the array buffer. After the Nth step, the elements are rearranged in sequency order. If during any step of the computation overflow occurs, the array is divided by two. The number of divisions is returned in a scale word.

Subroutine Calls:

RSTREG and SAVREG

Calling Sequence:

```
     MOV     #1$, R5
     JSR     PC, HADMAR
1$:  BR      .+10.
     .WORD   Address of a working buffer
     .WORD   Address of a location containing the number of elements in the
             array
     .WORD   Address of array to be transformed
     .WORD   Address of a scale word
```

The result is returned in the input array. The scale word contains the number of times that the array was divided by two.

All registers are not modified.

Program Name: HADXFM

Purpose:

To perform a Hadamard transform on a two-dimensional array.

Description:

This routine applies the one-dimensional Hadamard transform routine (HADMAR) to a two-dimensional array. This is accomplished by first applying it to the rows and then applying it to the resulting columns. The input array can either be an image or a Hadamard transformed image. The output is then a Hadamard transformed image or an image, respectively. When the output is a Hadamard transform, the option is provided to create an image which is a scaled version of the transform. A histogram of the transform is displayed on the graphics display to aid in selecting boundaries for scaling.

Subroutine Calls:

ALPHA, BLDISP, CHCUR, CLEAR, DSABLR, ENABLR, ERFAT, FPASCE, F.CL, F.C, F.C$, F.P, F.R, F.S, GRMODE, HADMAR, PLOT, RAD2AS, RSTREG, SAVREG, SNGDEC and TTYOUT

Calling Sequence:

Entry is from the Executive at:

HADXFM

Program Name: INVMAT

Purpose:

To invert a matrix.

Description:

This program inverts a square matrix of four-word floating elements. The result is returned in the same buffer.

Subroutine Calls:

GETIX, RSTREG and SAVREG

Calling Sequence:

```
     MOV      #1$, R5
     JSR      PC, INVMAT
1$:  BR       .+8.
     .WORD    Address of a location containing the address of the matrix
     .WORD    Address of a location containing the address of a working
              buffer
     .WORD    Address of a location containing the dimension of the matrix
```

General registers are not modified.

Program Name: KEYARA

Purpose:

To create a region file based on keyboard input.

Description:

This routine creates a region file based upon user keyboard entries. The entries can be the actual coordinate values or the upper left coordinate value of a rectangular region and its dimensions. A third option to enter cursor coordinates from an image display has been included. The current effort, however, does not include the display software.

Subroutine Calls:

ERFAT, ERREC, F.C, F.P, F.R, F.R$, F.S, OPTION, RSTREG, SAVREG

Routines that are not included in the current effort but that are required for cursor coordinate input are expected to have the following calling sequences:

Display an Image

JSR	R5,DISP1
BR	.+6
.WORD	Address of image file request block (file must not be open)
.WORD	Blow up factor (integer) Positive values blow up the image and negative values shrink it. Values of -1, 0 and 1 cause every image point to be displayed.

The following globals must be set prior to the call:

SROW	=	First row in image to be displayed
SCOL	=	First column in image to be displayed
OUTROW	=	First row on display to be used
OUTCOL	=	First column on display to be used

(KEYARA cont.)

Image Display Cursor

Enable Cursor

```
JSR         R5,ENABLC
BR          .+4
.WORD       Address of subroutine to call when coordinates are captured
            (when cursor interrupt occurs)
```

Disable Cursor

```
JSR         R5,DSABLC
```

Clear Image Display Graphics

```
JSR         R5,CLRGRF
```

Draw Lines on Display

```
JSR         R5,DRWLNE
BR          .+4
.WORD       Address of a parameter list
Parameter List:
.WORD       Row coordinate of point 1
.WORD       Column coordinate of point 1
.WORD       Row coordinate of point 2
.WORD       Column coordinate of point 2
```

Calling Sequence:

Entry is from the Executive at:

START

(KEYARA cont.)

The image display cursor routine calls the following subroutine when a coordinate is captured.

```
JSR         R5,TAKCRD
BR          .+6
.WORD       Row coordinate value
.WORD       Column coordinate value
```

Both row and column display values begin at zero, i.e., pixel at row 1, column 1 is displayed at display point 0,0.

Program Name: LINDEP

Purpose:

To find linear dependent rows of a matrix.

Description:

This routine generates a list of dependent rows in a specified square matrix. The matrix is expected to be in four-word floating point format.

Subroutine Calls:

GETIX, RSTREG and SAVREG

Calling Sequence:

```
     MOV     #1$, R5
     JSR     PC, LINDEP
1$:  BR      .+8
     .WORD   Address of a location containing the address of the matrix
     .WORD   Address of a location containing the address of a buffer in
             which to return the numbers of the dependent rows
     .WORD   Address of a location containing the dimension of the matrix
```

The return buffer consists of the number of dependent rows in the first byte and the numbers of the dependent rows in the remaining bytes.

Registers are not modified.

Program Name: LINES

Purpose:

To create a binary image where only the edges of objects in an input grey level image appear.

Description:

This routine finds the edge points of objects by comparing the averages of arrays of points surrounding each pixel. If the averages exceed a user specified threshold the output binary point is set at 255. If it does not exceed the threshold the output value is zero. The array averages are computed by calling the "SMOOTH" routine. The smoothed image is stored in a temporary file for processing.

A second option called "AREA EDGE DETECTION MAX" builds upon the first option. This routine considers neighboring row and column positions that have also exceeded the threshold. The position that exceeded the threshold by the greatest amount in the row is then set to 255. The same action is performed on the column.

Subroutines Called:

F.C, F.C, F.P, F.R, RAD2AS, SI2DA, F.SHUT, SMOOTH, SNGDEC, TTYIN and TTYOUT

Calling Sequence:

Entry is from the Executive at:

LINES - Area edge detection
MLINES - Area edge detection max.

Program Name: LNPRNT

Purpose:

To print specified character strings on the line printer.

Description:

This routine initializes and opens the line printer output file. Character strings terminated by a null are then accepted and listed in the line printer file.
A second calling point is available to close and spool the line printer file to the line printer.

Subroutines Called:

ERFAT, RSTREG and SAVREG

Calling Sequence:

To print one or more ASCII character strings:

```
     MOV       #1$, R5
     JSR       PC, LNPRNT
1$:  BR        2$:
     .WORD     Address of ASCII character string
     .WORD     Address of 2nd string (optional)
               .
               .
               .
     .WORD     Address of last string (optional)
2$:
```

To close and spool the file to the line printer:

```
     JSR       PC, LPCLOS
```

Registers are not modified.

Program Name: LOG

Purpose:

To compute the natural log or the common log of a number.

Description:

This routine computes the natural log of a floating point number. The method used is described by Hart.(1). The common log (base 10) is computed from the natural log by multiplying by the constant $Log_{10}(e)$.

Subroutines Called:

RSTFPS, RSTREG, SAVFPS, SAVREG

Calling Sequence:

```
     To compute the common log:
     MOV            #1$, R5
     JSR            PC, LOG
1$:  BR             .+6
     .WORD          Address of Executive Common
     .WORD          Address of Single Precision FP number

     To compute the natural log:
     MOV            #1$, R5
     JSR            PC, LN
1$:  BR             .+6
     .WORD          Address of Executive Common
     .WORD          Address of single Precision FP number
```

In both of the above cases, the result is returned in the subroutine argument. Upon an error, the flag BKIND is set and control is returned to the calling program.

General and floating point registers are not altered.

Program Name: LOGGER

Purpose:

To log messages and terminal I/0 into the Log File.

Description:

This routine logs text in a disk file named LOGFILIPS.TTN. The entry point 'LOGIT' allows for logging of a specified message, with time appended, into the Log File. The entry point 'LOGGER' will log interactive terminal I/0 in the Log File. Terminal input is logged with a 6 space indentation. Terminal output is logged with a 3 space indentation.

Subroutines Called:

ERREC, GETIM, RSTREG, SAVREG

Calling Sequence:

```
     For entry point LOGIT:
     MOV            #1$, R5
     JSR            PC, LOGIT
1$:  BR             .+10
     .WORD          Address of Executive Common
     .WORD          Address of message
     .WORD          Address of message length

     For entry point LOGGER:
     JSR            PC, LOGGER
```

where R1 contains the address of Executive Common.

General registers are not modified.

Program Name: LOGOVR

Purpose:

To start, stop or print the log.

Description:

This routine controls the operation of the automatic log routine 'LOGGER'. When the log is started, the log file 'LOGFILIPS.TTN' is created. 'N' in the file name will take on the value of the terminal number from which commands are being entered. When the log is stopped, the file may be spooled to the line printer by the user. The file is then deleted.

Subroutines Called:

ERREC, .PRINT, TTYIN and TTYOUT

Calling Sequence:

Entry is from the executive at:

LOGOVR

Program Name: LOGMOD

Purpose:

To provide a high level access to the logic file.

Description:

This routine contains subroutines to open, close and access the logic file. Included are routines to add, delete and retrieve logic tree nodes and class symbols.

Subroutines Called:

ERFAT, F.CL, F.E, F.P and F.R

Calling Sequence:

Open the logic file for multiple page read access:

```
     MOV       #1$, R5
     JSR       PC, OPLF
1$:  BR        .+8
     .WORD     Address containing the address of the I/O buffer
     .WORD     Address containing the length of the I/O buffer
     .WORD     Address of the logic file name
```

Upon return MAXCON contains the maximum number of pages that may be accessed at one time.

Open the logic file for single page read access:

```
     MOV       #1$, R5
     JSR       PC, OPENLF
1$:  BR        .+8
     .WORD     Address containing the address of the I/O buffer
     .WORD     Address containing the length of the I/O buffer
     .WORD     Address of the logic file name
```

Open the logic file for single page modify access:

```
     MOV       #1$, R5
     JSR       PC, OPLOGF
1$:  BR        .+8
     .WORD     Address containing the address of the I/O buffer
     .WORD     Address containing the length of the I/O buffer
     .WORD     Address of the logic file name
```

```
Close the logic file
     JSR       PC, CLLOGF
Close the logic file
     JSR       PC, CLLF
Set a pointer to one page of the opened logic file.
     MOV       #1$, R5
     JSR       PC, GTLCGF
1$:  BR        .+6
     .WORD     Address containing the append count
     .WORD     Address containing the page number to retrieve
```

The logic file may be opened by using any of the previously defined open subroutines. The append count is the number of pages to add to the file if the requested page does not physically exist.

Upon Return:

R3 contains the address of the logic page
FLOGP contains the first page number resident
LLOGP contains the last page number resident plus one

Set a pointer to the requested logic pages located in the opened logic file. (The logic file must be opened previously via a call to OPLF.):

```
     MOV       #1$, R5
     JSR       PC, GTLF
1$:  BR        .+6
     .WORD     Address containing the number of pages to read
     .WORD     Address containing the starting page number
```

Upon return:

R3 contains the address of the first logic page
FLOGP contains the first page number resident
LLOGP contains the last page number resident plus one

Add one level of nodes to a lowest node in the logic tree block of the opened logic file. (The logic file must be opened via a call to the subroutine OPLOGF.):

```
     MOV       R5,-(SP)
     MOV       #1$, R5
     JSR       PC, ADDNOD
1$:  BR        .+10
     .WORD     Address containing node number where nodes are to be added
     .WORD     Address containing the number of nodes on the new level
               (# of branches)
     .WORD     Address to transfer control upon error
     MOV       (SP)+, R5
```

The error return is executed if:

1. The specified node does not exist
2. The specified node is not a lowest node

Delete one level of logic nodes associated with a superior node from the logic tree block of the opened logic file. (The logic file must be opened via a call to OPLOGF.):

```
     MOV       R5,-(SP)
     MOV       #1$, R5
     JSR       PC, DELNOD
1$:  BR        .+6
     .WORD     Address containing the superior node number
     .WORD     Address to transfer control upon error
     MOV       (SP)+, R5
```

The error return is executed if:

1. There is more than one level of nodes below the superior node
2. The specified superior node is a lowest node or
3. The specified superior node does not exist.

Retrieve a node from the logic tree block of the opened logic file. (The logic file may be opened via any of the previously defined open subroutines, but if the node information is to be modified, the subroutine OPLOGF must be used to open the file.):

```
     MOV       R5,-(SP)
     MOV       #1$, R5
     JSR       PC, GETNOD
1$:  BR        .+6
     .WORD     Address containing the node number to retrieve
     .WORD     Address to transfer control if the node is inactive.
               (Error Return)
     MOV       (SP)+, R5
```

Upon Return:

R1 contains the address of the logic node

Upon Error Return:

If R1=0 the logic page is not defined for the node, otherwise the node is inactive.

Retrieve a class symbol entry from the class symbol block of the opened logic file. (The logic file may be opened via any of the previously defined open subroutines, but if the information is to be modified, the subroutine OPLOGF must be used to open the logic file.):

```
     MOV       R5,-(SP)
     MOV       #1$, R5
     JSR       PC, GETSYM
1$:  BR        .+6
     .WORD     Address containing the node number entry
     .WORD     Address to transfer control upon error
     MOV       (SP)+, R5
```

LOGMOD (Continued)

Upon Return:

R2 contains the address of the second byte in the entry, i.e. the number of symbols. The class symbols follow in successive bytes.

Upon Error Return:

An entry associated with the requested node does not exist in the class symbol block.

Delete a class symbol entry from the class symbol block of the opened logic file. (The logic file must be opened via a call to OPLOGF.):

```
     MOV       R5, -(SP)
     MOV       #1$, R5
     JSR       PC, DELSYM
1$:  BR        .+6
     .WORD     Address containing the node number of the entry
     .WORD     Address to transfer control upon error
     MOV  (SP)+, R5
```

An error occurs if the subroutine fails to find an entry associated with the node number located in the parameter list.

Add a symbol entry to the class symbol block of the opened logic file. (The logic file must be opened via a call to OPLOGF.):

```
     MOV       R5,-(SP)
     MOV       #1$, R5
     JSR       PC, ADDSYM
1$:  BR        .+6
     .WORD     Address containing the address of the class symbol
               entry
     .WORD     Address to transfer control upon error
     MOV       (SP)+, R5
```

The class symbol entry is described as follows:

Byte	Description
0	Node number
1	Number of symbols
2	First symbol (ASCII)
	.
	.
	.
	Last Symbol

An error occurs if there already exists an entry in the class symbol logic block for the node.

Program Name: LPHDCP

Purpose:

To list grey value or binary images on the line printer.

Description:

This routine lists grey value or binary images on the line printer. A three digit number is printed for each pixel value within the user selected portion of the image. For binary images binary 1 is printed as an asterisk and a binary 0 as a blank. If the number of columns is greater than the width of the line printer, the output is generated in several sections.

Subroutines Called:

ERREC, F.CL, F.P, F.R, LNPRNT, LPCLOS, RAD2AS, SI2DA, SI2DAR, SNGDEC, and TTYOUT

Calling Sequence:

Entry is from the executive at:

EDGPRN - List binary image
GRYDMP - List grey value image

Program Name: MDOTV

Purpose:

To compute the dot product between a matrix and a vector.

Description:

This routine forms the dot product between a matrix and a vector. The elements of each must be in floating point. Single or double precision values are accepted. The processor must be set to the desired precision prior to entry. The column dimension of the matrix must equal the dimension of the vector.

Subroutines Called:

GETIX, RSTREG and SAVREG

Calling Sequence:

```
     MOV       #1$, R5
     JSR       PC, MDOTV
1$:  BR        .+12
     .WORD     Address of the address of the matrix
     .WORD     Address of the address of the vector
     .WORD     Address of the address for the result
     .WORD     Address of the number of columns in the matrix
     .WORD     Address of the number of rows in the matrix
```

Registers AC0, AC1 and AC3 are modified.

Program Name: MENCOV

Purpose:

To compute the mean vector and covariance matrix.

Description:

The mean vector and covariance matrix is computed for each class in the current data set. Only those vectors which are located at the current logic tree node will be used for the calculations. The output is stored in the file "MCFILE.MC". If the file exists, it is deleted and recreated.

Subroutines Called:

CLLOFG, CLOVEC, ERFAT, F.CL, F.C, F.D, F.P, GETSYM, GETVEC, LOADER, OPENLF, OPNVEC, RSTREG, SAVREG and SNGDEC.

Calling Sequence:

Entry is at:

MENCOV

Program Name: MESAGE

Purpose:

To generate the file ERRMESFIL.IPS

Description:

This routine converts the recoverable error message file ERRMESREC.SRC and the fatal error message file ERRMESFAT.SRC to the IPS acceptable format in file ERRMESFIL.IPS. The two input files may be changed using the text editor.

Subroutines Called:

None

Calling Sequence:

This routine is executed via the RSX RUN command. Entry is at:

MESAGE

Program Name: MODREP

Purpose:

To remove image noise points.

Description:

This routine determines the modal value of the pixels within a 3 by 3 neighborhood of each image point. If the center point differs in value by more than a specified amount, it is replaced with the modal value.

Subroutines Called:

ERFAT, F.C, F.P, F.R, F.S, RAD2AS, RSTREG and SAVREG

Calling Sequence:

Entry is from the Executive at:

MODREP

Program Name: NORMLZ

Purpose:

To normalize an image.

Description:

The normalization process is performed in one of two ways. Either the image grey values are expanded to the full 0 to 255 range or the values are normalized over a user specified range. The function performed is dependent upon the entry point.

Subroutines Called:

ERREC, F.C, F.P, F.R, F.S, RAD2AS, RSTREG, SAVREG, SI2DA and SNGDEC

Calling Sequence:

Entry is from the Executive at:

NORMLZ - Normalize over 0 - 255 range
RNGCHG - Normalize over the user specified range.

Program Name: ODDLD

Purpose:

To remove noise points or lines from an image.

Description:

This routine performs one of the two noise removal functions based upon the entry point selected. Both operations, however, make their decisions based upon the state of the 3 by 3 neighborhood about each pixel. The odd dot entry point searches for pixels that differ from one or more or the average of their eight neighbors. The "differ" criterion and the number of neighbors are user specified.. Those pixels which satisfy the condition are replaced with the average of those neighbors with which they differ.

The odd line entry point functions in a slightly different manner. The point is replaced with the average only if two neighbors which themselves are adjacent cannot be found that are within the user-specified threshold.

Subroutines Called:

ERREC, F.C, F.P, F.R, F.S, RAD2AS, SI2DA, SNGDEC, TTYIN

Calling Sequence:

Entry is from the Executive at:

ODDLIN - Odd line entry
ODDDOT - Odd dot entry

Program Name: OPTION

Purpose:

To display option lists and accept selections.

Description:

This routine simply lists a specified option list (not frame option lists) on the graphics display and accepts the user's input. The return address is then selected on the basis of this input. The number of options is determined by the length of the parameter list.

Subroutines Called:

ERREC and SNGDEC

Calling Sequence:

```
     MOV          R5, -(SP)
     MOV          #1$, R5
     JSR          PC, OPTION
1$:  BR           2$
     .WORD        EXBUF - Address of Executive Common
     .WORD        Address of Prompt
     .WORD        Address of Length of Prompt
     .WORD        Address of Option 1
     .WORD        Address of Option 2
     .
     .
     .
     .WORD        Address of Option N
2$:  MOV          (SP)+, R5
```

R5 must be pushed onto the stack prior to the JSR, so that the "option" subroutine is compatible with the "call" MACRO.

Program Name: PARCOR

Purpose:

To partition the free core buffer.

Description:

This routine divides the Free Core Buffer into a number of fractional parts. the addresses and sizes of these parts are then placed directly into request blocks.

Subroutines Called

SAVREG and RSTREG

Calling Sequence:

```
     MOV          #1$, R5
     JSR          PC, PARCOR
1$:  BR           2$
     .WORD        EXBUF - Address of Executive Common
     .WORD        Denominator of Fraction
     .WORD        Numerator 1
     .WORD        ADR of BUF ADR Parameter in File Req Block
     .WORD        Numerator 2
     .WORD        ADR of BUF ADR Parameter in File Req Block
     .
     .
     .
     .WORD        Numerator N
     .WORD        ADR of BUF ADR Parameter in File Req Block
2$:
```

Registers are not modified.

Program Name: PLOT

Purpose:

To provide routines for generating graphic plots on the Tektronix display.

Description:

Several subroutines are contained within this program which together allow graphic plots to be easily generated.

Subroutines Called:

ERFAT, ERREC, RSTREG, SAVREG and TTYGRF

Calling Sequence:

To select alpha mode:

```
     MOV            #1$, R5
     JSR            PC, ALPHA
1$:  BR             .+4
     .word          Address of Executive Common
```

To select graphics mode:

```
     MOV            #1$, R5
     JSR            PC, GRMODE
1$:  BR             .+4
     .WORD          Address of Executive Common
```

To clear screen:

```
     MOV            #1$, R5
     JSR            PC, CLEAR
1$:  BR             .+4
     .WORD          Address of Executive Common
```

The display is left in alpha mode with the cursor at the upper left margin.

To position the cursor at the "home" position:

```
     MOV            #1$, R5
     JSR            PC, HOME
1$:  .WORD               .+4
     .WORD          Address of Executive Common
```

The display is left in graphics mode.

(PLOT cont.)

To plot a vector:

```
     MOV            #1$, R5
     JSR            PC, PLOT
1$:  BR             .+8
     .WORD          Address of Executive Common
     .WORD          Address of X coordinate
     .WORD          Address of Y coordinate
```

The routine draws a dark vector if the call is immediately preceeded by a call (PLOT cont.)

Program Name: PTEDGE

Purpose:

To find object edges in grey value images.

Description:

This routine is designed to find points which are the edges of objects. This is accomplished by comparing each point to user specified neighbor points. If the difference exceeds a user specified threshold, the output point is set to 255. Otherwise the output value is 0.

Subroutines Called:

ERFAT, F.C, F.P, F.R, F.S, RAD2AS, SI2DA and SNGDEC

Calling Sequence:

Entry is from the Executive at:

PTEDGE

Program Name: PWLINR

Purpose:

To accept and apply user-specified piecewise-linear transfer functions.

Description:

This routine allows the user to specify the transfer function by entering coordinates at the keyboard. These coordinates describe the end points of the linear sections. The transfer function can be saved in a file for future use.

The newly specified transfer function or a previously existing one can be applied to an image via this routine.

Subroutines Called:

ALPHA, BLDISP, CLEAR, ERREC, F.C, F.R, F.P, F.S, GRMODE, OPTION, PARCOR, PLOT, RAD2AS, SNGDEC, TRNFTN, TTYGRF, TTYIN, and TTYOUT

Calling Sequence:

Entry is from the Executive at:

PWLINR

Program Name: RAD2AS

Purpose:

This routine converts a string of RAD50 packed characters to an ASCII string.

Subroutines Called:

RSTREG and SAVREG

Calling Sequence:

```
     MOV        #1$, R5
     JSR        PC, RAD2AS
1$:  BR         .+10
     .WORD      Address of RAD50 string
     .WORD      Address for ASCII ouptut
     .WORD      Address of number of words in RAD50 string
```

Program Name: RAMDSP

Purpose: To output a grey level image to the RAMTEK Display.

Description: The user enters the starting row and column of the image and the blow-up factor. The image file is accessed and the DMA transfer to the RAMTEK begins. Upon completion, the file is closed and control returns to the executive.

Subroutines Called:

F.CL, F.P, F.R, RSTREG, SAVREG, SNDGEC, TTYOUT

Calling Sequence:

Entry is from the executive at:

RAMDSP

NOTE: This option is not fully implemented.

Program Name: RENAME

Purpose:

To rename a file

Description:

This routine allows a file to be renamed. The old file name and new name are obtained from the user. No change in the extension is allowed.

Subroutines Called:

ERFAT, ERREC and GETNAM

Calling Sequence:

Entry is from the Executive at:

RENAME

Program Name: RGSTAT

Purpose:

To compute and list region statistics.

Description:

This routine computes and lists on the graphics display or line printer the mean, standard deviation, mode, and population of a selected region of an image file or sets of regions and images as described by a spectral set file.

Subroutines Called:

BEGFND, BLDISP, CFREQ, CSDEV, CLEAR, ERFAT, ERREC, F.CL, F.P, FINDPT, GMEAN, GSDEV, GMEDN, GMODE, LNPRNT, RAD2AS, RSTREG, SAVREG, SI2DA, SMINIT, SNGDEC, TTYGRF, TTYIN.

Calling Sequence:

Entry is from the Executive at:

RGSTAT - Operate on spectral set

SPSTAT - Operate on single region/image pair

Program Name: RSTRT

Purpose:

Restart a Frame Task

Description:

This routine restarts a Frame Task. The stack pointer is reset, the display address is reset to the BLDISP routine, and the display is rebuilt if necessary. The routine then jumps to the main control loop.

Subroutines Called:

ALPHA, BLDISP, CTLOOP

Calling Sequence:

The routine is entered following a Frame Task startup or a fatal error by executing a

```
JMP             RSTRTT
```

R4 must contain the address of Executive Common

Program Name: RTIOMG

Purpose:

To compute the ratios of two images on a point by point basis.

Description:

This routine makes two passes through the file. On the first pass, the range of values is obtained by multiplying each image point to be normalized by 256 and dividing the result by the reference image point. On the second pass, these values are then normalized between 0 and 255.

Subroutines Called:

ERFAT, F.C, F.P, F.R, F.S, PARCOR, RAD2AS, and TTYOUT

Calling Sequence:

Entry is from the Executive at:

RTIOMG

Program Name: SARFPS

Purpose:

To save and restore the status of the floating point processor.

Description:

The FPP's registers and status register are saved on and restored from the stack:

Subroutines Called:

None

Calling Sequence:

```
To save the FPP status:
JSR             PC, SAVFPS
To restore the FPP status:
JSR             PC, RSTFPS
```

Program Name: SAVER

Purpose:

To save and restore the general registers R0 - R5

Description:

The general registers R0 - R5 are saved on and restored from the stack.

Subroutines Called:

None

Calling Sequence:

```
To save registers:
JSR             PC, SAVREG
To restore registers:
JSR             PC, RSTREG
```

Program Name: SELNOD

Purpose:

To obtain the node number from the user and to call the appropriate logic routine.

Description:

This routine requests the desired node number from the user and stores it in the global location "LOGNOD". Control is then passed to the logic creation routine associated with the entry point chosen.

Subroutines Called:

ERREC and SNGDEC

Calling Sequence:

Entry is from the Executive at:

SELNOD - Fisher logic

SELBOL - Boolean logic

Program Name: SELOGF

Purpose:

To retrieve or create a logic file.

Description:

The existence of the current vector set is checked as described by global "VECTNM." A fatal error is reported if none exists. The logic file name is requested next. If the file exists, the class symbols in the vector file set and the logic file are compared to insure compatibility. The vectors are all reset to node 1 in the logic tree. The logic evaluation routine is then called to apply any existing logic to the vectors.

If the file does not exist, a new one is created. The class symbols are extracted from the vector file and placed in the logic file.

Subroutines Called:

ADDSYM, CLLOFG, CLOVEC, CRLOG, ERFAT, F.CH, GETNM, GETSYM, GETVEC, GTLOGF, LOADER, OPENLF, OPNVEC, RSTREG, SAVREG, and TTYOUT.

Calling Sequence:

Entry is at:

SELOGF

Program Name: SELVEC

Purpose:

To obtain the vector set file name for future logic operations.

Description:

This routine requests the vector set file name from the user and stores it in the global variable "VECTNM". The vector set is opened to obtain the vector dimension, the vector measurement format indicator and the vector header format indicator. These are stored in global locations "VECDIM", "VECMFI" and "VECHFI" respectively.

Subroutines Called:

GETNM, OPNVEC and CLOVEC

Calling Sequence:

Entry is at:

SELVEC

Program Name: SHIFTI

Purpose:

To shift an image vertically and horizontally.

Description:

This routine shifts an image by a user specified row and column amount. The shift is accomplished by a circular rotation of the rows and columns in the image.

Subroutines Called:

ERREC, F.C, F.P, F.R, F.S, PARCOR, RAD2AS, SI2DA and SNGDEC

Calling Sequence:

Entry is from the Executive at:

SHIFTI

Program Name: SIMSMO

Purpose:

To obtain a user specified box size and then call the smoothing routine.

Description:

This routine obtains the input and output files and the dimensions of the smoothing array. These parameters are then passed to program "SMOOTH".

Subroutines Called:

BYTDEC, F.C, F.R, F.S, RAD2AS, SI2DA, SMOOTH and TTYOUT

Calling Sequence:

Entry is from the Executive at:

SIMSMO - M by N smoothing array

BOXSMO - Square smoothing array

Program Name: SMOOTH

Purpose:

To smooth an image.

Description:

The smoothing operation is performed by computing the average value of the points in a specified neighborhood about each point and replacing the center point with this value. Edge points are effectively extended outward to satisfy "missing" neighborhood points for edge points.

Subroutines Called:

ERFAT, F.P, RSTREG and SAVREG

Calling Sequence:

```
      MOV     #1$,R5
      JSR     PC,SMOOTH
1$:   BR      2$
    .WORD     Address of input file request block
    .WORD     Address of output file request block
    .WORD     Address of number of rows in smoothing array
    .WORD     Address of number of columns in smoothing array
2$:
```

The files are assumed to be open.

Program Name: SPAMEZ

Purpose:

To compute the mean, variance, standard deviation, median, mode, high, low and range of a set of numbers.

Description:

The statistics are computed on a frequency table that is constructed by this routine. The frequency table is the number of times that each grey value occurred in the data passed to "CFREQ".

Subroutines Called:

RSTFPS, RSTREG, SAVREG, SAVFPS and SQRT

Calling Sequence:

To initialize the frequency buffer, the sum and the sum of squares prior to calling CFREQ, CMEAN and CVARI:

```
          MOV       #1$,R5

          JSR       PC,SMINIT

1$:       BR        2$

          .WORD     Address of data value (byte value at an even location)
                    to be supplied by CFREQ, CMEAN or CVARI.  Also this is
                    the address where the computed values are returned from
                    GMEAN, GVARI, GSDEV, GMEDN, GMODE, GHIGH, GLOW and GRANGE

          .WORD     Address of a 256 word buffer for the frequency table.

          .WORD     Address of the number of data values.

          .WORD     Address of mode flag.  This is an optional parameter.
                    It must be supplied if GMODE is used.

2$:

                    The mode flag is set by GMODE to the number of modes
                    minus one.
```

To tally a byte of data into the frequency table:

```
          JSR       PC,CFREQ
```

To add data to the current sum:

```
          JSR       PC,CMEAN
```

Program Name: SPAMEZ

To add data to the sum and the square of data to the sum of squares:

```
JSR     PC,CVARI (or CSDEV)
```

To calculate the mean of the data:

```
JSR     PC,GMEAN
```

CMEAN must be called first

To calculate the variance of the data:

```
JSR     PC,GVARI
```

CVARI must be called first

To calculate the standard deviation of the data:

```
JSR     PC,GSDEV
```

CVARI must be called first

To calculate the median of the data:

```
JSR     PC,GMEDN
```

CFREQ must be called first.

To calculate the mode of the data:

```
JSR     PC,GMODE
```

CFREQ must be called first.

To calculate the high data value:

```
JSR     PC,HIGH
```

CFREQ must be called first.

To calculate the low data value:

```
JSR     PC,LOW
```

CFREQ must be called first

To calculate the range of values:

```
JSR     PC,RANGE
```

CFREQ must be called first

Program Name: SPBIAS

Purpose:

To add a constant to an image.

Description:

This routine simply adds a user specified constant to each point in an image.

Subroutine Calls:

DI2DAR, F.C, F.P, F.R, F.S, PARCOR, RAD2AS, SI2DA, SNGDEC and TTYOUT

Calling Sequence:

Entry is from the Executive at:

SPBIAS

Program Name: SQRT

Purpose:

To compute the square root of a number.

Description:

This routine computes the square root of a double or single precision integer. The value is computed using Newton's method with an initial guess of (2**15)-1.

Subroutine Calls:

RSTREG and SAVREG

Calling Sequence:

```
     Double precision argument:
     MOV            #1$, R5
     JSR            PC, DPSQRT
1$:  BR             .+4
     .WORD          Address of the double precision integer (low order part first)
```

The square root is returned in the low order word of the argument.

```
     Single precision argument:
     MOV            #1$, R5
     JSR            PC, SQRT
1$:  BR             .+4
     .WORD          Address of single word integer
```

The square root is returned in place of the argument.

Registers are not modified.

Program Name: STRTSK

Purpose:

To request execution of a new Frame Task.

Description:

This routine requests the RSX-11M Executive to initiate the new Frame Task specified by the argument list. The task name is constructed using the two ASCII characters supplied, prefixed by an F and followed by a TTn where n represents the terminal number of the initiating task. The six character name is converted to RAD50 and requested. If the task cannot be initiated, a recoverable error is declared and a return is executed. If the request is for the currently active task, a jump is executed to the task restart point. When a new task is successfully started, the log file is closed if it is active and the calling task is terminated.

Subroutines Called:

AS2RAD, ERREC, RSTREG, SAVREG

Calling Sequence:

```
     MOV       #1$, R5
     JSR       PC, STRTSK
1$:  BR        .+6
     .WORD     Address of Executive Common
     .WORD     Address of 2 ASCII characters to be used in building the
               task name (may be a byte boundary).  A leading 0 is required
               for single digit Frame numbers.
```

Program Name: TAPE

Purpose:

To perform various tape operations.

Description:

This routine reads and writes image files from and to magnetic tape. The images are stored on tape as one row per record. The last record is followed by an end-of-file mark. Record size is limited to the range 18 to 2048 bytes.

A skip file option and a rewind option are also available.

Subroutine Calls:

ERFAT, ERREC, F.CL, F.C, F.P, F.R, SNGDEC and TTYOUT

Calling Sequence:

Entry is from the Executive at:

TIN	-	Read tape file
TOUT	-	Write tape file
SKIPF	-	Skip tape file
REWF	-	Rewind tape

Program Name: TEKDSP

Purpose:

To display a binary image on the Tektronix display

Description:

This routine displays a binary image on the Tektronix where binary "one" points are displayed as a dot (period) and binary "zero" points are displayed as a blank position. The user can specify that an array of dots be used for a point and he can also specify the spacing between dots. Both have a "blow up" effect.

Subroutine Calls:

ALPHA, BLDISP, CLEAR, ERREC, F.P, F.R, F.S, GRMODE, HOME, PLOT, RAD2AS, SI2DA, SNGDEC, TTYIN, and TTYOUT

Calling Sequence:

Entry is from the Executive at:

TEKDSP

Program Name: TELEIO

Purpose:

To provide graphics display I/O.

Description:

This routine allows character strings to be printed on the graphics display and to be accepted from the keyboard. The input routine first outputs a prompt and then accepts input from the keyboard.

The rebuilding of the option frames is also controlled by this routine.

Subroutine Calls:

ERFAT, ERREC, LOGGER, RSTREG, and SAVREG, BLDISP

Calling Sequence:

To output a character string with appended carriage return and line feed:

```
     MOV        #1$, R5
     JSR        PC, TTYOUT
1$:  BR         .+8.
     .WORD      Address of Executive Common
     .WORD      Address of output string
     .WORD      Address of length of output string
```

To input a character string

```
     MOV        #1$, R5
     JSR        PC, TTYIN
1$:  BR         .+8.
     .WORD      Address of Executive Common
     .WORD      Address of operator prompt
     .WORD      Address of length of prompt
```

The input string is returned in the TTYBF in the Executive Common, and the string length is stored at offset IOSLN relative to the base of the Common.

If an "ALT MODE" key is struck, the equivalent of a total error is executed.

To output a graphics character string:

```
     MOV            #1$, R5
     JSR            PC, TTYGRF
1$:  BR             .+8.
     .WORD          Address of Executive Common
     .WORD          Address of output string
     .WORD          Address of length of string
```

To input the position of the graphics cursor:

```
     MOV            #1$, R5
     JSR            PC, TTGRIN
1$:  BR             .+4
     .WORD          Address of Executive Common
```

The five byte cursor data string is returned in TTYBF$ in the Executive Common.

Display rebuilding is enabled when the flag REBLD$ in Executive Common is set.

Program Name: TMPARA

Purpose:

To report that the image display options are not implemented.

Description:

This routine simply prints "OPTION INOPERATIVE!!!" when called. Control is then returned to the executive.

Subroutine Calls:

TTYOUT

Calling Sequence:

```
JSR          PC,DISP1(ENABLC, DSABLC, CLRGRF or DRWLNE)
```

Program Name: TRNFTN

Purpose:

To apply an image transfer function.

Description:

This routine applies a specified transfer function to an image. This function is in the form of a 256 byte look-up table. The input image grey value is used as an index into this table to determine the output value.

Subroutine Calls:

F.P, RSTREG and SAVREG

Calling Sequence:

```
     MOV          #1$, R5
     JSR          PC, TRNFTN
1$:  BR           .+8.
     .WORD        Address of input file request block
     .WORD        Address of output file request block
     .WORD        Address of transfer function in memory.
     Registers are not modified.
```

Program Name: TTYIO

Purpose:

To provide high level terminal I/O.

Description:

This routine provides terminal I/O functions that request parameters from the user and accept his response. These parameters include file specifications and numerical input (octal and decimal). Floating point numerical input is accepted.

Subroutine Calls:

ERFAT, ERREC, RSTFPS, RSTREG, SAVFPS, SAVREG, TTYIN and TTYOUT

Calling Sequence:

Output a message and input a file specification (the file name is checked for legality)

```
     MOV          #1$, R5
     JSR          PC, GETNAM
1$:  BR           +12.
     .WORD        Address of Executive Common
     .WORD        Address of message
     .WORD        Address of length of message
     .WORD        Address of file request block
     .WORD        Address of blank line indicator
```

The file name is input from the keyboard and parsed using the RSX-11M command string interpreter. The parsed file name information is returned in the file request block specified. If the input cannot be parsed, the input request is repeated.

Output a message and input a string of signed double word integer decimal numbers.

```
     MOV        #1$, R5
     JSR        PC, DBLDEC
1$:  BR         Offset to end of parameter list
     .WORD      Address of Executive Common
     .WORD      Address of prompt
     .WORD      Address of length of prompt
     .WORD      Address of location containing the maximum count of numbers
                to input.  This count must be satisfied exactly or an error
                message is printed and a request is made to retype the line.
     .WORD      Address of array for numbers.  This array must contain
                sufficient space to store all the numbers indicated by the
                second parameter.  The numbers are entered on the keyboard
                separated by commas and preceeded by a minus or plus sign.
                The absence of a sign is interpreted as a plus sign.  Zero
                is returned for a given number if its corresponding field
                is empty (comma only).
     .WORD      Address of location containing CTRL-Z indicator (optional
                argument)  When specified this argument indicates that the
                line terminator is not a CTRL-Z, a zero will be returned
                in the indicator.  If it is a CTRL-Z, then a -1 is returned.
```

Output a message and input a string of signed single word integer decimal numbers.

```
     MOV        #1$, R5
     JSR        PC, SNGDEC
1$:  BR         Offset to end of parameter list
                (The parameter list and description is identical to DBLDEC
                except that single word values are returned.)
```

Output a message and input a string of signed single byte decimal numbers.

```
     MOV        #1$,R5
     JSR        PC, BYTDEC
1$:  BR         Offset to end of parameter list
                (The parameter list and description is identical to DBLDEC
                 except that single byte values are returned)
```

Output a message and input a string of signed double word integer octal numbers.

```
     MOV        #1$, R5
     JSR        PC, DBLOCT
1$:  BR         Offset to end of parameter list
                (The parameter list and description is identical to DBLDEC.)
```

Output a message and input a string of signed single word integer octal numbers.

```
     MOV        #1$, R5
     JSR        PC, SNGOCT
1$:  BR         Offset to end of parameter list
                (The parameter list and description is identical to DBLDEC
                except that single word values are returned.)
```

Output a message and input a string of signed single byte integer octal numbers.

```
     MOV        #1$, R5
     JSR        PC, BYTOCT
1$:  BR         Offset to end of parameter list
                (The parameter list and description is identical to DBLDEC
                except that single byte values are returned.)
```

Output a message and input a floating point string of numbers. These numbers can be in "F" or "E" format.

```
     MOV        #1$, R5
     JSR        PC, FLTENT
1$:  BR         Offset to end of parameter list
                (The parameter list and description is identical to DBLDEC
                except single precision floating point values are returned.)
```

General and FPP registers are not modified.

Program Name: VECFNS

Purpose:

To provide sequential access to vectors in a vector set.

Description:

This routine provides services to open a set of vector files listed in a vector set file, to access the files and to close the files. The access is on a vector by vector basis. The vectors are returned sequentially. Only one vector file is open at a time.

Subroutine Calls:

ERFAT, F.CL, F.P, F.R, F.R$, RSTREG and SAVREG

Calling Sequence:

```
     Open vector set:
     MOV       #1$, R5
     JSR       PC, OPNVEC
1$:  BR        .+12.
     .WORD     Address of the address of an I/O buffer
     .WORD     Address of the length of the buffer
     .WORD     Address of a four word vector set file specification (device,
               unit, and filename)
     .WORD     Address of location containing open access (=read, 1=read/write)
     .WORD     Address of a three word buffer for return parameters.
               Word 0 = Vector dimensionality
               Word 1 = Measurement format indicator (from file header)
               Word 2 = Vector header format indicator (from file header)
     Retrieve next sequential vector:
     MOV       #1$, R5
     JSR       PC, GETVEC
1$:  BR        .+4
     .WORD     Return address when no further vectors remain
```

The memory address of the vector is returned in R0.

Close vector set:

```
JSR        PC, CLOVEC
```

Program Name: WTDSMO

Purpose:

To perform a weighted smooth.

Description:

This routine performs the smooth by operating on a user specified neighborhood of each point. Each neighbor is multiplied by a weight which is stored in a weight array. This array is equal in dimension to that of the neighborhood. The sum of the products is then divided by the sum of the weights, the sum of the weight absolute values or a user-entered number. The form of this divisor is specified by the user.

Subroutine Calls:

BYTDEC, ERFAT, F.C, F.P, F.R, F.S, OPTION, RAD2AS, SI2DAR, SNGDEC, TTYIN, and TTYOUT

Calling Sequence:

Entry is from the Executive at:

WTSMO - Weighted smooth

ABSWS - Absolute value weighted smooth

APPENDIX A

IPS SUPPORT FILES

The Image Processing System has support files associated with it. These files aid in the building of the Executable system. The files provide three essential functions:

1. Assemble all source code into object libraries.
2. Build all the frame tasks.
3. Start-up the Image Processing System.

The description of each of these support files follows.

(NOTE: In all examples, the set UIC is assumed to be that which contains all of the IPS files.)

Filename: DEVICE.CMD

Function: Task builds DEVICE common block.

Issuing Format: TKB @ SR:DEVICE

Description:

The following devices are used:

SR: - support command file

SY: - task image and symbol table of Device

Filename: FRAMES.TSK

Function: To build the IPS compatible master option list file from the text Edited master option list file.

Issuing Format: RUN TK:FRAMES

Description:

The file MSTROPTON.SRC must be present on the system device SY. This file may be modified using the text editor. The Program "FRAMES" will convert this file to an IPS compatible file called MSTROPTON.LST.

The source code FRAMES.MAC must be assembled and then the object file task built to produce FRAMES.TSK.

Filename: FRMXX.CMD

Function: Task build a single frame which is determined by the frame number "XX" in the filename. (Example: FRM01.CMD)

Issuing Format: TKB @ SR:FRMXX.CMD

Description:

The following pseudo devices are used:

SR: - support command files

LY: - object library IPS.OLB

TK: - frame task

MP: - frame memory allocation map
(Spooled to LP:)

Filename: FRMXX.ODL

Function: Overlay descriptor file for a single frame.

Issuing Format: None. It is referenced by the FRMXX.CMD file only.

Description:

The following Pseudo device is used:

LY: - object library IPS.OLB

Filename: IPSBLD.CMD

Function: Task builds IPS start-up task.

Issuing Format: TKB @ SR:IPSBLD

Description:

The following Pseudo devices are used:

SR: - support command files
LY: - object library IPS.OLB and IPSRES.OLB
OB: - object module IPS.OBJ
SY: - symbol table [1,54] RSX11M.STB
MP: - memory allocation map (spooled to LP:)
TK: - task IPS.TSK

The source code IPS.MAC is assembled into an object module as follows:

```
MAC OB:IPS, LI:IPS = SY:[1,1]EXEMC/ML, SR: [11,2] MACS, COMMON, IPS
```

Filename: IPS.CMD

Function: Creates library IPS.OLB

Issuing Format: @SR:IPS

Description:

The following Pseudo devices are used:

SR: - support command files and source files
OB: - object files (temporary storage only)
LI: - assembly listing (spooled to LP:)
LY: - object library IPS.OLB

Filename: IPSRES.CMD

Function: Creates resident library IPSRES.OLB

Issuing Format: @SR:IPSRES

Description:

The following Pseudo devices are used:

SR: - support command files and source files
OB: - object files (temporary storage only)
LI: - assembly listing (spooled to LP:)
LY: - object library IPSRES.OLB

Filename: IPSTASK.CMD

Function: Task Builds frames 1 to 13

Issuing Format: @SR:IPSTASK

Description:

The following Pseudo devices are used:

SR: - support command files
LY: - object library IPS.OLB
TK: - all frame tasks
MP: - all frame memory allocation maps. (spooled to LP:)

Filename: IPSRES.TKB

Function: Task builds resident library IPSRES.OLB

Issuing Format: TKB @SR:IPSRES.TKB

Description:

The following devices are used:

SR: - support command file
SY: - task image and symbol table of IPSRES

Filename: MESAGE.TSK

Function: To build the IPS compatible Error file from Text edited error files

Issuing Format: RUN TK:MESAGE

Description:

The files ERRMESREC.SRC and ERRMESFAT.SRC must be present on the system device SY:. These two files may be modified using the Text Editor. The program "MESAGE" will merge these two files into one IPS compatible file called ERRMESFIL.IPS.

The source code MESAGE.MAC must be assembled and then the object file task built to produce MESAGE.TSK.

Filename: SETIPS.CMD

Function: Allows user to assign Pseudo devices to Physical devices.

Issuing Format: @SR:SETIPS

Description:

The following Pseudo device is used:

SR: - support command files

Filename: SETUP.TTO

Function: installs all uninstalled frames and assigns the task name FXXTTO.

Issuing Format: @SR:SETUP.TTO

Description:

This command file assumes that the TEKTRONIX DISPLAY TERMINAL is physical device TTO:

The following devices are used:

SR: - support command files

APPENDIX B

IPS START-UP PROCEDURE

Before the system may be run, the following must be established.

1. The UIC must be set to that which the system is stored under.
2. The start-up task "IPS" must be installed.
3. Library and device partitions must be created and installed.
4. The Frame tasks must be resident on device TK:.
5. The Resident library, Device partition, Error file, and master option file must be resident on the system device SY:.

Items 1 through 4 above are established at system boot time via the command file "STARTUP.CMD". The specific commands are

```
SET/UIC = [11,2]
INS IPS/TASK=IPSTO
SET /MAIN = IPSRES:1260:400:COM
INS IPSRES
SET IMAIN = DEVICE :7600:200:DEV
INS DEVICE
ASN DK1:=TK:
```

The above command list assumes all IPS software is on directory [11,2] and that all frame tasks are on device DK1:. If that is not the case, the "SET" and "ASN" commands should be changed appropriately.

The following files must be resident on the system device:

```
IPSRES.TSK
DEVICE.TSK
ERRMESFIL.IPS
MSTROPTON.LST
SETUP.TTO
```

This start-up sequence assumes that the Tektronix Terminal is device TTO:. If this is not the case, the files "STARTUP.CMD" and "SETUP.TTO" must be modified.

APPENDIX C

SOURCE ASSEMBLY PROCEDURE

There are three types of source modules. They are:

1. Frame modules
2. Overlay modules
3. Library modules

Frame modules form the base to each frame task. They are assembled as follows (with the exception of Frame 12):

```
MAC FRMXX = MACS, COMMON, FRMXX
```

"MACS" is the Macro file and "COMMON" is a common data block.

Frame 12 is assembled differently since the upper 4K words of the task are mapped into the "DEVICE" partition. Frame 12 is assembled as follows:

```
MAC FRM12 = MACS,COM12,FRM12
```

The Frame object modules are stored in the library "IPS" as follows:

```
LBR IPS/IN = FRMXX/-EP
```

Overlay modules form the functional part of each frame (such as option overlays). These modules are assembled as follows:

```
MAC OVERLAY = MACS, OVERLAY
```

They are inserted into the "IPS" library as follows:

```
LBR IPS/IN=OVERLAY
```

Library modules are commonly referenced subroutines. They are written with Re-entrant and position independent code. They are assembled as follows:

```
MAC LIB = MACS, COMSYM, LIB
```

They are inserted into the resident library "IPSRES" as follows:

```
LBR IPSRES/RP = LIB
```

APPENDIX D

ERROR MESSAGES

The system error file ERRMESFIL.IPS contains the error messages for all error conditions that might occur during normal system execution. This file contains 100 fixed length records of 40 bytes each. The first 50 records are recoverable errors. The last 50 records are fatal errors. The record number is directly related to the error number to provide for random access of the error messages. The relationship is as follows:

For recoverable errors:

Record Number = Recoverable Error Number + 1

For fatal errors:

Record Number = Fatal Error Number + 51

The file ERRMESFIL.IPS is created by the IPS support program MESAGE which obtains the error messages from ERRMESREC.SRC and ERRMESFAT.SRC. These two files consist of variable length records and hence is compatible with the Text Editor.

RECOVERABLE ERROR MESSAGES - ERRMESREC.SRC

E0.0 SPECIF. FILE CANNOT BE CREATED
E0.1 FILE HAS INCORRECT DATA TYPE
E0.2 INCORRECT FILE SPECIF. FORMAT
E0.3 INPUT STRING SYNTAX ERROR
E0.4 ILLEGAL FRAME NUMBER
E0.5 ILLEGAL OPTION NUMBER
E0.6 INPUT STRING TOO LONG
E0.7 PREVIOUS PREMATURE FILE CLOSE
E0.8 REF. PICTURE ELEMENT NONEXISTANT
E0.9 DUPLICATE FILE NAME
E0.10 INCORRECT PARAMETER VALUE
E0.11 NONEXISTANT PICT. ELEMENT REF.
E0.12 INCORRECT NUMBER OF ENTRIES
E0.13 INCORRECT FILE FORMAT
E0.14 ILLEGAL FILE NAME
E0.15 SPECIF. FILE IS NONEXISTANT
E0.16 ILLEGAL DEVICE
E0.17 INCORRECT
E0.18 TOO MANY PARAMETERS
E0.19 ILLEGAL CHARACTER
E0.20 PARENTHESES DO NOT BALANCE
E0.21 ILLEGAL OPERATOR
E0.22 OPERAND MISSING
E0.23 ARITHMETIC OPERAND EXPECTED
E0.24 LOGICAL OPERAND EXPECTED
E0.25 OPERATOR MISSING
E0.26 ARITH. STATEMENT INCOMPLETE
E0.27 LOGICAL ARGUMENT INCOMPLETE
E0.28 ILLEGAL OR INCOMP. STATEMENT
E0.29 INSUFFICIENT NUMBER OF POINTS
E0.30 FLOATING POINT OVER/UNDER FLOW
E0.31 ILLEGAL SYMBOL
E0.32 ILLEGAL LIMITS
E0.33 ILLEGAL MEASUREMENT
E0.34 DID NOT USE ALL CLASS SYM.
E0.35 LOG FILE ACCESS ERROR

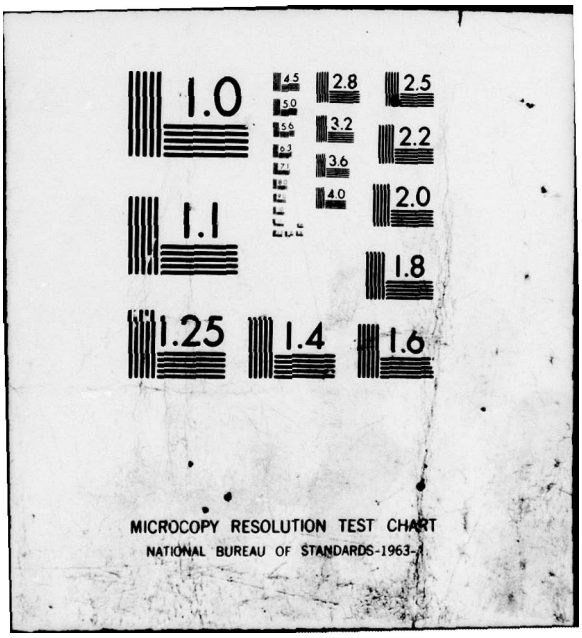
FATAL ERROR MESSAGES - ERRMESFAT.SRC

E1.0 HARDWARE ERROR ON TRANSFER
E1.1 CORE SPACE EXHAUSTED
E1.2 FILE DIMENSION ERROR
E1.3 DUPLICATE FILE NAME
E1.4 ILLEGAL DEVICE
E1.5 DEVICE FULL
E1.6 TEMPORARY FILE ERROR
E1.7 SYSTEM TABLE OVERFLOW
E1.8 BAD UID OR DIRECTORY FULL
E1.9 DISPLAY COORDINATES ILLEGAL
E1.10 NONEXISTANT RECORD REFERENCE
E1.11 CORE SPACE EXHAUSTED
E1.12 PROTECTION CODE VIOLATION
E1.13 FILE OPEN
E1.14 LENGTH OR NO. OF RECORDS ZERO
E1.15 IMAGE DIMENSION INSUFFICIENT
E1.16 ZERO ENTRIES
E1.17 ILLEGAL FILE NAME
E1.18 A REQUIRED FILE IS NONEXIS.
E1.19 INCORRECT FILE FORMAT
E1.20 FILE HAS INCORRECT DATA TYPE
E1.21 RECORD SEQUENCE ERROR
E1.22 LOG FILE CREATE ERROR
E1.23 LOG FILE OPEN ERROR
E1.24 FAILURE ON "OPEN" OPERATION
E1.25 INCORRECT NO. DEP. VARIABLES
E1.26 LOG ARG. OR = TO 0
E1.27 NEIGHBOR OPERATIONS ILLEGAL
E1.28 UNEXPECTED END OF FILE
E1.29 ILLEGAL LOGIC FILE OPERATION
E1.30 CLASS SYMBOLS INCONSISTENT
E1.31 NODE DOES NOT EXIST
E1.32 NODE NOT LOWEST NODE
E1.33 SYMBOLS ALREADY EXIST AT NODE
E1.34 ILLEGAL REGION
E1.35 MULTIPLE SYMBOLS AT A LOW NODE
E1.36 NO CLASS SYMBOLS IN VECTOR FILE
E1.37 LP SPOOLER FILE ACCESS ERROR
E1.38 MEASUREMENT IS NONEXISTANT

AD-A073 653 AMHERST SYSTEMS INC BUFFALO NY F/G 9/2
IMAGE PROCESSING SYSTEM SOFTWARE. VOLUME II. PROGRAMMING MANUAL--ETC(U)
JUN 79 E G EBERL, P T GLINSKI F30602-78-C-0077
UNCLASSIFIED AMHERST-0077-VOL-2 RADC-TR-79-52-VOL-2 NL

3 OF 3
AD A073653

END
DATE FILMED
10-79
DDC


1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
4.5
5.0
5.6
6.3
MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

APPENDIX E

DATA FILE DESCRIPTIONS

Several different file formats are used within the system to store various data types. Each type is identified by a different filename extension. The descriptions of each type are found on the following pages.

Certain items in the file headers are common to all types. These items are found in the first and last 64 words of the header. The first 64 words are as follows:

Word	Description
0	Reserved for future use
1	Filename extension (RAD50)
2	Total number of records in file
3	Record length in bytes
4	File close indicator 0 = File was normally closed 1 = File was prematurely closed
5-63	Reserved for future use

The last 64 words of the header are used to store the descriptive text. This text can contain embedded carriage returns and line feeds and is terminated with a null.

File Type: Function File

Extension: .FCT

Special Header Parameters:

Word	Description
64	Set to negative one to show non-array file
65	Function indicator: set to 1 for one-dimensional function. Set to 2 for two-dimensional function.

Data Format:

This file type is used to store the compiled code associated with a filter function. The entire buffer returned by the compiler routine "COMPIL" is stored as one record in this file. Consult "COMPIL" documentation for detailed information. The file format follows (only one record in file):

Word	Description
0	Number of words in this record
1	Offset to compiled code relative to word 0
2	Offset to symbol table relative to word 0
3	Offset to symbol values relative to word 0
4	Filter function in the original ASCII string format
	.
	.
	.
	Symbol table
	.
	.
	.
	Compiled Code
	.
	.
	.
	Symbol values

File Type: Filter File

Extension: .FLT

Special Header Parameters:

Word	Description
64	Array format indicator. Set to 3 to show two-word floating point.

Data Format:

This file type is used to store filter arrays. The elements of the array are in two-word floating point format. One row of elements is stored in one record. The number of records is equal to the number of rows in the filter array.

File Type: Hadamard Transform File

Extension: .HAD

Special Header Parameters:

Word	Description
64	Array Element Format Indicator 1 = Word Elements 2 = Double Word Elements

Data Format:

This file type is used to store Hadamard transform arrays. The elements of the array are as indicated above. One row of elements is stored in one record. The number of records is equal to the number of rows in the Hadamard array.

File Type: Image File

Extension: .IMG

Special Header Parameters:

Word	Description
64	Set to zero to indicate integer byte array elements.

Data Format:

This file type is used to store images. One row of picture elements is stored in one record. The number of records is equal to the number of rows in the image.

File Type: Logic Tree

Extension: .LOG

Special Header Parameters: None

Data Format:

This file type is used to store logic trees and their associated logic. Each record in the file is called a page where record 1 is page 0. The records are always 512 bytes in length. The formats of the various page types are detailed on the following pages.

Directory Page - Page 0 (Rec. 1)

Word	Description
0	Number of active entries in directory
1	Number of currently active pages in file
Entry 1: 2	Low byte: Logic tree node number if entry is associated with logic. This is set to zero for non-logic entries
	High byte: Entry identifier: 1 = One-Space Logic (to be added) 2 = Two-Space Logic (to be added) 3 = Fisher Pairwise Logic 4 = Boolean Logic 5 = Reserved for future logic type. 6 = Logic Tree 7 = Class symbols
3	Page number of block described by this entry
4	Number of pages in this block
	.
	.
	.
	Entries continue in the same format as entry 1.

Page 1 (Rec. 2) Reserved for Future Use

Logic Tree Block - Page 2 (Rec. 3)

(Additional Logic Tree Pages are created as required.)

Byte	Description
0	Current page number
1	Number of active entries
2-9	Reserved for future use
10-509	Fifty, ten byte node entries *
510-511	Not used

* Node Entry Format

Byte	Description
0	Node number. Zero if entry is inactive
1	Node level in tree
2-5	Reserved for future use
6	Node number of immediate senior node
7	Number of nodes on next level below this node. Zero indicates that the current node is a terminal node.
8	Node number of first node below
9	Node number of the next node on the current level which has the same immediate senior node. Zero indicates that the current node is the last node on this level.

Note: Node numbers begin at 1 and are assigned consecutively as requested by the logic creation routines.

Node levels also begin at 1 with the senior node and increase consecutively at each lower level.

Class Symbol Block - Page 3 (Rec. 4)

(Additional Class Symbol pages are created as required)

Byte	Description
0	Number of entries on this page
1-511	Class Symbol Entries.* Each entry is of variable length. There is no space between entries.

* Class Symbol Entry Format

Byte	Description
0	Node number in logic tree with which the symbols are associated.
1	Number of symbols in this entry
2	1st class symbol
3	2nd class symbol
	.
	.
	.
	Last class symbol

Fisher Pairwise Logic

A set of Fisher logic corresponding to one node in the logic tree may occupy several consecutive logic file pages. The logic consists of a header block which is stored in one or more pages and the logic components which are also stored in one or more pages. In both cases when extending beyond a page, continuation takes place at the beginning of the next page. The logic components begin at the top of the first page after the header.

Header Format

Word		Description
0,1		Reserved for future use
2		Number of bytes in this header (always even)
3		Offset from top of header to the general information region
4		Offset from top of header to the class symbol region
5		Offset from top of header to the criterion description region
6		Offset from top of header to the measurement description region
7	General Information Region	Minimum vote count
8		Number of the highest dimension with which the logic was created (not necessarily the dimensionality of the vectors)
9,10		Reserved for future use
11	Class Symbol Region	Number of classes
12		Low Byte: Node number to which the class is associated
.		
.		
.		
		High Byte: Class symbol
		.
		.
		.
		A one word entry appears for each class plus one extra entry for the reject category.

(Fisher Pairwise Logic cont.)

Measurement Description Region	Number of measurements used. Measurement numbers used (one byte per entry) . . .
Criterion Description Region	Low Byte: Logic type. This entry allows logic other than Fisher to be used for a given class pair. This is currently set to 3 to indicate Fisher logic High Byte: Number of thresholds used. The current version only uses the distance between the class means. Therefore, this is set to 1. Page number of logic relative to the first page of the header Number of pages in logic Relative byte of the start of the logic within the page . . . (The above four word entry format is repeated for each class pair in the set of classes on which the logic was created.

Fisher Criterion Format

Word		Description
0	FV_1	Fisher vector in two-word floating point format. NDIM is the largest dimension with which the logic was created. This is not necessarily the dimensionality of the vectors.
.	FV_2	
.	.	
.	.	
	.	
	FV_{NDIM}	
	DV_1	This is the discriminant vector for the above Fisher vector. This is not currently used, but is stored for future use.
	DV_2	
	.	
	.	
	.	
	DV_{NDIM}	
	θ_1	Two-word floating point threshold values.*
	θ_2	
	θ_3	
	θ_4	
	θ_5	
		.
		.
		.
		(The above is repeated for each class pair.)

* Five thresholds are computed and stored for each class pair. Only threshold three is currently used by logic evaluation. The other thresholds are stored for possible future software additions. The thresholds are computed as follows:

Let μ_i be the estimated mean of class i projected onto the Fisher direction d, i.e.,

$$\mu_i = \underset{\sim}{d}^T \underset{\sim}{\bar{X}}_i \qquad i = 1,2$$

where $\bar{X}_i$ is the mean of class i.

Then the thresholds are:

$$\theta_1 = \mu_2 - \Delta/2$$

$$\theta_2 = \mu_2 + \Delta/3$$

(Fisher Criterion Format cont.)

$$\theta_3 = \mu_2 + \Delta/2$$

$$\theta_4 = \mu_1 - \Delta/3$$

$$\theta_5 = \mu_1 + \Delta/2$$

where

$$\Delta = \mu_1 - \mu_2$$

Boolean Logic Format

Word	Description
0,1	Reserved for future use
2	Number of bytes in logic
3	Low Byte: Node number of false node High Byte: Node number of true node
4	Address of compiled code relative to word 0
5	Address of symbol table relative to word 0
6	Address of symbol values relative to word 0
7	Input ASCII character string terminated with a null
	.
	.
	.
	Symbol Table
	.
	.
	.
	Compiled Code
	.
	.
	.
	Symbol Values

Symbol Table Format

Word	Definition
0	Number of symbols
2	Status word for symbol 1
3,4	RAD50 pack of symbol 1
5	Status word for symbol 2
6,7	RAD50 pack of symbol 2
	.
	.
	.
	Status word for last symbol
	RAD50 pack of last symbol

Status Word Format

Low Byte:	This is always set to 1. Other values returned by the compiler are not accepted by the logic creation routine. (See COMPIL for other values.)
High Byte:	Set to the measurement number (binary) which the symbol represents.

Calling the Compiled Code

The compiled code is called as follows:

```
JSR        PC,(adr. of code)
```

Prior to this call, the values of all measurements for the current vector must be provided. These values are placed in the buffer area identified by word 6 of the Boolean logic. The first four words of this area are reserved for special subroutine addresses. The first three should be filled with the entry point addresses of the EXP, LN and LOG routines respectively. The fourth is a special routine for image neighborhood operations. This feature of the compiler will not occur in the logic. Therefore, this word is not used.

Following the four special addresses appear the measurement values in the order of appearance within the symbol table. All values are in two-word floating point. The logical result is returned in R1 as 1 for true or a 0 for false.

File Type: Mean-Covariance File

Extension: .MC

Special Header Parameters: None

Data Format:

This file type is used to store the mean vectors and covariance matrices from which Fisher logic is created. One record exists in the file for each class. The record format follows:

Word	Description
0	Class symbol in lower byte. Upper byte unused.
1,2	Two-word floating point count of the number of vectors used in computing the mean and covariance
3	MV MV^1 MV^2 . . . MV $NDIM$ — Two-word floating point mean vector. NDIM is the number of dimensions in the vector set.
	Upper Triangle of covariance matrix This is stored by row beginning with the diagonal element.

File Type: Region File

Extension: .REG

Special Header Parameter: None

Data Format:

This file type is used to store region boundary description. The coordinate pairs of the vertices are stored one each in a four byte record. Each record contains the row coordinate value in the first two bytes and the column coordinate value in the last two bytes. The last record contains the same coordinate pair as the first record. This forces a closed boundary.

File Type: Spectral Set File

Extension: .SPC

Special Header Parameters:

Word	Description
64	Always 0
65	Number of image file names
66	Number of region file names

Date Format:

This file type is used to store the description of a set of spectral images. Also included are names of region files from which vectors can be created. The record length is 14 bytes where each record contains an image file specification or one region file specification. Image file specifications appear first, followed by the region file specifications. The formats are as follows:

Spectral Set File Record Format

Image File Names:

Word	Description
0	Device name (RAD50)
1	Unit No.
2-3	File Name (RAD50)
4	Extension (RAD50)
5	User ID
6	Not Used
7	Not Used

Region File Names:

Word	Description
0	Device Name (RAD50)
1	Unit No.
2-3	File Name (RAD50)
4	Extension (RAD50)
5	User ID
6	Class Symbol
7	Data Reduction Factor

File Type: Vector Set File

Extension: .VCS

Special Header Parameters: None

Data Format:

This file type is used to store the description of a set of vector files. The record length is 14 bytes where each record contains one vector file specification as follows:

Word	Description
0	Device Name (RAD50)
1	Unit No.
2-3	File Name (RAD50)
4	Extension (RAD50)
5	User ID
6	Not Used
7	Not Used

File Type: Vector File

Extension: .VEC

Special Header Parameters:

Word	Description
64	Meas. Format Ind. 0 = Integer Byte 1 = Integer Word 2 = (Not used) 3 = Flt. Pt. (Single precision) 4 = (Not used)
65	Vector Header Format Indicator Bit 0: 0 = Perm. and Temp. Sym. Space not Allocated 1 = Perm and Temp. Sym. Space Allocated Bit 1: 0 = Row and Column not present 1 = Row and Column present
66	Total number of meas. per vector
67	Number of vectors per record
68-69	Total number of vectors
70	Row dimension of original image
71	Column dimension of original image
72	Vector length in bytes (always even)
73	Number of spectral measurements

Data Format:

This file type is used to store measurement vectors. Each vector file corresponds to one image file from which the vectors were extracted. Depending upon the total number of vectors, one or more vectors may be stored in a record. This is done to keep the record count to single precision. When more than one vector is stored per record, the records are packed with no intervening space. The format of the vector is as follows:

Word	Description
0	M_1
.	M_2
	.
	.
	.
	M_N — Measurements (see header word 64 for format)
	Low Byte: Node number
	High Byte: Not used
	Low Byte: Temporary class symbol
	High Byte: Permanent class symbol
	Row number in image at which vector was found
	Column number in image at which vector was found

APPENDIX F

EXECUTIVE OPTION REQUIREMENTS

The following executive services are required by the Image Processing System. These options must be included at system generation time.

1. User oriented terminal driver
   (specifically IO.RPR, TF.BIN, TF.RAL, TF.RNE, IO.WAL).
2. Alter priority directive.
3. Get task parameters directive.
4. Get partition parameters directive.
5. Memory management directives.
6. Checkpointing.
7. Dynamic memory allocation/compaction.
8. 8K common partition reserved for library.
9. Remaining memory in system controlled partition.
10. ANSI magtape support.

APPENDIX G

TASK BUILD PROCEDURE

There are four types of tasks in the system. They are:

1. Resident Library Task
2. Frame Tasks
3. Start-up Task
4. Device Partition Task

The resident library task and symbol table must exist before the frame tasks may be built. The resident library partition must exist, then the following command will build the task:

```
TKB @ SR:IPSRES.TKB
```

All 13 frame tasks may be built with the following command:

```
@SR:IPSTASK
```

The start-up task is called IPS and is assembled as follows:

```
MAC OB:IPS, LI:IPS=[1,1]EXEMC/ML,SR:[11,2]MACS,COMMON,IPS
```

It is task built as follows:

```
TKB @SR:IPSBLD
```

The device task maps the upper 4K words of frame 12 into the I/O page. The device task is built as follows:

```
TKB @SR:DEVICE
```

MISSION
of
Rome Air Development Center

RADC plans and executes research, development, test and selected acquisition programs in support of Command, Control Communications and Intelligence (C^3I) activities. Technical and engineering support within areas of technical competence is provided to ESD Program Offices (POs) and other ESD elements. The principal technical mission areas are communications, electromagnetic guidance and control, surveillance of ground and aerospace objects, intelligence data collection and handling, information system technology, ionospheric propagation, solid state sciences, microwave physics and electronic reliability, maintainability and compatibility.